Designing the Data Architecture of a Logistics Delivery Company

De Vera, Janelle Gloriane R.,
Garcia, Felipe Jr. D.,
Obiso, Bernadette M.,
Verdan, James Patrick T.

Data Engineering Final Project

Asian Institute of Management
Aboitiz School of Innovation, Technology and Entrepreneurship
Master of Science in Data Science

21 September 2023

Table of Contents

List of Figures

List of Tables

Introduction

Background of the Study

WZP WZP Delivery Services, Inc. is a logistics company delivering daily packages of various types and sizes across the Philippines. It started in 2000 with a small-scale operation in Metro Manila. It grew to serve the entire nation and even have international partners. It has 69 delivery hubs scattered across different provinces, with expansion already on the horizon. It serves different clients, such as business-to-business (B2B) and business-to-customer (B2C).

The company is undergoing significant change, primarily due to the launch of its digitalization program, in response to the impact of the COVID-19 pandemic on the business. Delivery riders can directly save data and information about the packages they delivered through an app called TracKING. Raw data is stored in the cloud and accessed by the company through data extraction that outputs a CSV file.

Problem Statement

In the company's operations, an important pre-delivery activity is manpower prediction. The delivery riders are not part of the company's employees but are provided by a third-party services provider (TPSP). The number of riders employed is subject to weekly negotiations between the company and the third-party service provider (TPSP). Therefore, the WZP WZP must accurately predict the manpower for delivering all the packages in their custody. Similarly, employing descriptive analytics on packages will yield valuable insights into the types of packages the delivery hub anticipates receiving. Armed with this knowledge, delivery hubs can make more informed decisions about how to efficiently group packages based on their characteristics and the available workforce.

The study aims to create a data architecture that helps solve these manpower predictions and package batching challenges, from the seamless raw data preprocessing to loading it to data lakes and warehouses.

Significance of the Study

The Courier, Express, and Parcel (CEP) market in the Philippines to which the company belongs to, has a size of USD 1.62 Billion in 2023 and is expected to grow to USD 2.44 Billion by 2023 at a rate of 8.57% (CAGR).[1] The CEP market is closely tied to other markets, including e-commerce. With the COVID-19 pandemic lockdown, the increase in e-commerce further fueled the boom in the CEP market.

The Philippine CEP market is more closely described as fragmented rather than consolidated. This means that the industry is highly competitive and has no dominant players. The current market leaders are DHL, FedEx, UPS, LBC Express, and Philippine Postal Corporation.[1] This fragmentation also means new players in the market can easily snatch the market share through innovation. Thus, there is a compelling necessity to establish a robust data architecture within WZP WZP to remain competitive in the market. This data infrastructure will facilitate more efficient delivery operations through data-driven strategies and insights.

Scope and Limitation

The project scope encompasses the entire data lifecycle, starting from raw data extraction and extending through the preprocessing phase, making the data readily accessible for consumption on dashboards. This comprehensive approach aims to support descriptive and predictive analytics seamlessly. The architectural solutions will be dedicated exclusively to the logistics operation's last-mile activity, which is the delivery hub.

The project is constrained to the adoption of Amazon Web Services (AWS) and Apache products. While the design principles may be adaptable to other service providers, the actual implementation demonstrated in this study will be grounded in utilizing these two specific technologies.

Methodology

Design Considerations

Raw data comes from the input of delivery riders from their mobile applications. The raw data tracks the status of each package, whether it is pending, successful, or failed. This data includes tracking package statuses, indicating whether they are pending, successfully delivered, or failed. The preprocessed data will then be used in predictive analytics (manpower forecasting) and/or descriptive analytics. The architecture must streamline how raw data will be processed and stored. Various users in the organization will then use these stored data with varying purposes and goals (like Data scientists or Data Engineers).

Data Architecture

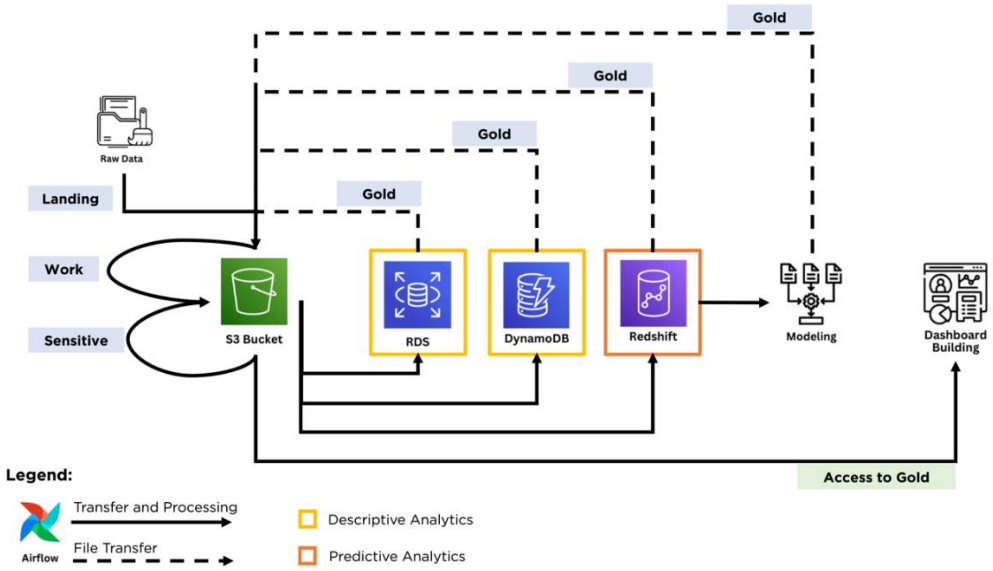


Figure 1. Data Architecture.

Data Architecture Overview

The data architecture above has four main components: *S3 Bucket, Relational Database Service (RDS), Redshift, and DynamoDB.* The data architecture above has four main components: S3 Bucket, Relational Database Service (RDS), Redshift, and DynamoDB. To streamline and optimize data operations, Apache Airflow was utilized. Airflow orchestrates tasks and workflows across these components, ensuring seamless data processing and management, enhancing efficiency in the data ecosystem.

The process starts by extracting the *Raw Data* from the TracKING app and other data sources before saving it in the Landing Zone of the *S3 bucket*. The preprocessed data, sensitive data, and transformed data are also stored under the S3 bucket in their respective zones. The full details of the data flow are discussed in the *Data Lake* section of this paper.

From the *S3 bucket*, the process then splits into three distinct processing and transformation components: (1) *RDS,* (2) *DynamoDB, and* (3) *Redshift.* Each component is selected based on the type of data being processed and its intended use. Both *RDS* and *DynamoDB* will handle data for Descriptive Analytics, while *Redshift* will handle data for Predictive Analytics. After processing and transformation, the processed data is then transferred to the Gold Zone in the S3 Bucket.

Among these three components, *Redshift* serves as the gateway for role-specific access to transformed data, enabling modeling to predict manpower requirements. Subsequently, the processed data is transitioned to the Gold Zone.

Lastly, all data under the Gold Zone can be visualized in the *Dashboard Building* component for analysts to generate further insights.

Design Justification

The data architecture effectively aligns with the core design principles, centered on optimizing the processing and storage of raw data originating from the TracKING mobile application and various other sources. It empowers predictive and descriptive analytics while accommodating the unique needs of diverse user roles throughout the organization. The efficiency consideration is justified by the deliberate separation of data processing and transformation into three distinct components. Each component is designed for different purposes and is tailored to handle specific data types, optimizing their processing performance. For example, structured data is processed more efficiently in RDS, while semi-structured data is better suited for DynamoDB, and Redshift excels in handling complex analytical queries.

Furthermore, this separation of data processing contributes to efficiency and scalability. The architecture allows for the independent scaling of each component, ensuring that resources are allocated precisely where needed. By isolating workloads, resource contention between different types of data processing is prevented. This isolation enhances overall system scalability and ensures that one component's demands do not impact the performance of others.

In addition to efficiency and scalability, the architecture also allows for flexibility. Each component possesses the ability to accommodate various data models and structures. This flexibility empowers the handling of diverse data formats with ease. As data changes over time, each component's flexibility in handling its data type allows for easier adaptation without major architectural overhauls.

Component Selection Justification

Using *Amazon S3 (Simple Storage Service)* offers various benefits, such as its cost-effectiveness, data versioning, and data security. One of the remarkable attributes of this architecture is its scalability and elasticity, capable of virtually accommodating extensive volumes and a wide variety of data types. Its inherent flexibility enables the organization and categorization of data into discrete zones within the Data Lake, establishing an efficient and structured data architecture.

The deliberate separation of the three processing and transformation components is selected mainly for their optimal ability to handle specific data types. *RDS* is typically used for structured, tabular data, making it suitable for storing data related to delivery information, delivery hub information, client information, package information, and recipient information. These datasets often have a relational structure, and RDS is well-suited for such data models. Leveraging its robust SQL querying capabilities, RDS allows for efficiently performing ad-hoc queries, generating reports, and extracting valuable insights from structured data. Additionally, by centralizing the related data in RDS, the architecture enables seamless data integration, facilitating joins, aggregations, and the resolution of complex analytical inquiries across various data sources.

DynamoDB's key advantage is its adaptability to semi-structured data, accommodating evolving data structures without rigid schemas. Its high throughput and low latency capabilities are crucial for real-time information, ensuring swift access and updates. Additionally, its scalability ensures DynamoDB grows seamlessly with expanding delivery operations.

Redshift excels in managing large analytical workloads and complex queries with its columnar storage and Massively Parallel Processing (MPP) architecture. Chosen for predictive analytics, Redshift's query optimization and performance power support intricate statistical models and machine learning algorithms. Separating workloads isolates resource-intensive tasks, maintaining query performance for modeling.

Results and Discussions

Data Lake

The purpose of the data lake in this project is to democratize data. It aims to provide self-service to various users while capturing data across different sources. The data lake for this use case has four zones: landing zone, work zone, gold zone, and sensitive zone, each with specific access and restrictions. The access to each zone will depend on the type of user (role). In the organization, there are four types of roles a data user may assume: Data Scientists, Data Engineers, Data Stewards, and Business Analysts. The purpose of each zone and role are discussed below.

Data Lake Zones

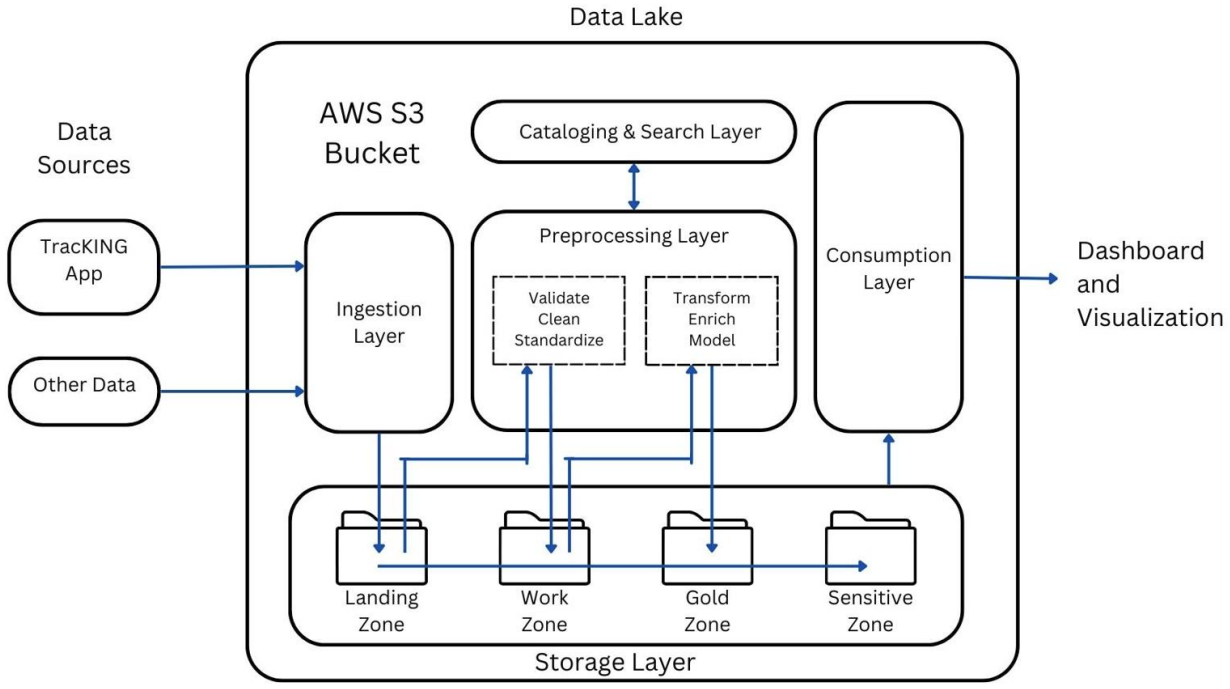


Figure 2. Data Lake with All Four Storage Layer Zones.

Note: This section only contains an in-depth description, for the full implementation, please see the appendix.

The data starts by extracting the necessary data from the TracKING app and saving it in the Landing Zone of the S3 bucket. The data is then preprocessed by validation, cleaning, and standardizing, after which it is then passed to the Work Zone. The Work Zone contains cleaned data already preprocessed for possible pipelines, such as exploratory data analysis. The Work Zone data is transformed, enriched, and modeled to be stored in the Gold Zone. Here, the data are ready for data modeling and other data science and business analysis purposes. Throughout this process, whenever any sensitive data is encountered, such as customer addresses or any other personally identifiable information (PII), it is first transferred to the Sensitive Zone. The four data lake zones are implemented as the main directories in an AWS S3 bucket.

Access Control

Roles	Read Access				Write Access			
	Landing Zone	*Work Zone*	*Gold Zone*	*Sensitive Zone*	*Landing Zone*	*Work Zone*	*Gold Zone*	*Sensitive Zone*
Data Scientists	NO	YES	YES	YES	NO	NO	YES	NO
Data Engineers	YES	YES	NO	NO	YES	YES	NO	NO
Data Stewards	YES	YES	YES	YES	YES	NO	YES	YES
Business Analysts	NO	YES	YES	YES	NO	NO	NO	NO

Figure 3. Access Control of Different Roles to the Data Lake Zones.

The access control is determined based on the user's role given the four zones. Four different roles in the organization can access different parts of the data lake. This provides them the necessary autonomy to conduct data writing/reading, all within the confines of their roles. They can view the zone's content for all roles, but the writing and reading of data depends on the roles. Data Scientists have read access to work, gold, and sensitive data to support them in their modeling projects. They also have write access in the gold zone to save progress or processed data. Data engineers' main role is to ensure proper data ingestion and preprocessing; therefore, they have both read and write access in landing and work zones. On the other hand, data stewards are responsible for ensuring all these data and restrictions are followed. They have read and write access on all zones except write access on the work zone. Finally, the business analyst doesn't have any write access but concerns themselves with reading the preprocessing and enriching data in the work and gold zone to be analyzed and visualized in the dashboard and reports. They also have read access to sensitive zones when necessary. The summary of the read-and-write access of each role is summarized in the table below.

Online Transaction Processing (OLTP)

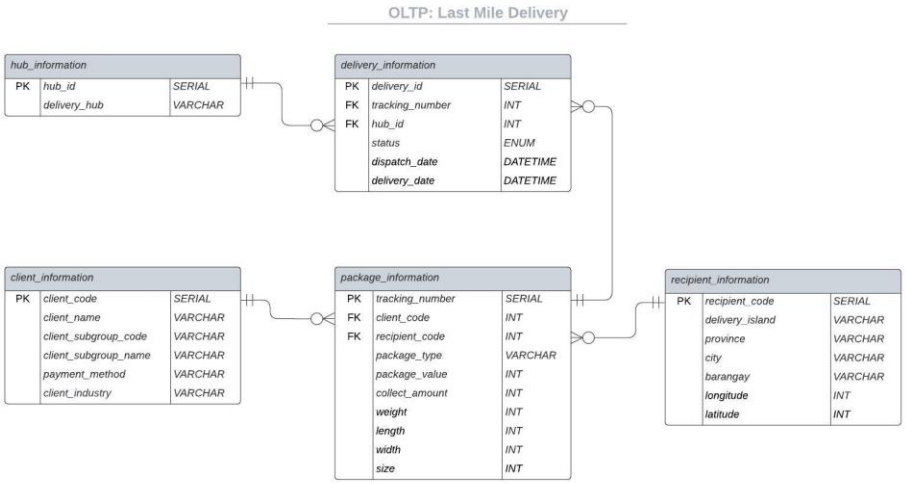


Figure 4. OLTP for Package Delivery Database.

Note: This section only contains an in-depth description, for the full implementation, please see the appendix.

The Online Transaction Processing schema tailored for last-mile delivery operations is the foundation for descriptive analytics within WZP WZP Delivery Services, Inc. It is meticulously designed to capture, process, and manage transactional data related to the final phase of product delivery, commonly called the "last mile." This schema orchestrates the interaction between key tables to optimize last-mile delivery operations.

At its core, the "*delivery_information*" table tracks delivery progress, while the "*hub_information*" table identifies the hub where packages are located. The "*client_information"* table holds client details, and the "*package_information*" table offers insights into package specifics. The "*recipient_information*" table includes essential information for riders to complete deliveries. These tables collectively drive operational efficiency, facilitate performance evaluations, assist in resource allocation, elevate the customer experience, and enable data-driven decision-making.

The schema's motivation stems from the logistics sector's quest for efficiency and customer satisfaction. Streamlining last-mile operations and enhancing customer interactions aims to position WZP WZP Delivery Services, Inc. as a competitive force in the ever-evolving logistics industry.

Table 1. List of Features in Delivery Information in OLTP Table.

Column Name	Key Type	Description
delivery_id	Primary Key	Serves as a unique identifier for each individual delivery within the system. It distinguishes one delivery from another and facilitates precise tracking and management of delivery records.
tracking_number	Foreign Key	Contains a unique alphanumeric code assigned to each package or shipment. This code allows tracking and monitoring of the package's location and status throughout delivery. This is from the package_information table.
hub_id	Foreign Key	Stores an identifier that corresponds to the specific hub or distribution center responsible for managing the package. It helps in routing and organizing packages efficiently for timely delivery. This is from the hub_information table.
status	-	Indicates the current state or condition of a delivery or package. It provides information on whether a delivery is pending or successfully delivered.
dispatch_date	-	Records the date and time when a delivery is dispatched from the hub or distribution center. It marks the beginning of the delivery journey and is crucial for tracking and monitoring delivery progress.
delivery_date	-	Captures the date and time when a package is successfully delivered to its intended recipient. It signifies the completion of the delivery process and is essential for recording delivery timelines and customer notifications.

Table 2. List of Features in Hub Information in OLTP Table.

Column Name	Key Type	Description
hub_id	Primary Key	Serves as a unique identifier associated with each delivery hub or distribution center within the WZP WZP network. It is used to distinguish and manage different hubs efficiently, allowing for precise tracking of packages' origins and destinations.
delivery_hub		Provides a descriptive name or label for each delivery hub or distribution center. It offers a human-readable identifier that helps identify and refer to specific hubs within the logistics network. This column enhances the user-friendliness of hub-related information.

Table 3. List of Features in Client Information in OLTP Table.

Column Name	Key Type	Description
client_code	Primary Key	It contains a unique code or identifier assigned to each client or customer of the organization. It serves as a reference code for easy identification and management of client accounts.
client_name	-	Stores the full name or title of each client or customer. It provides the formal name by which the client is known and is useful for clear client identification and communication.
client_subgroup_name	-	Categorizes clients into subgroups based on specific criteria or characteristics. Example: grocery, dress, debit.
payment_method	-	Indicates the method or mode of payment preferred or used by each client. It includes details on whether the client pays via credit card, bank transfer, cash, or any other specified method.
client_industry	-	Records the industry or sector to which each client or customer belongs. It categorizes clients based on business type, facilitating market segmentation and industry-specific analysis.

Table 4. List of Features in Package Information in OLTP Table.

Column Name	Key Type	Description
tracking_number	Primary Key	Contains a unique numeric code assigned to each package or shipment. This code allows for precise tracking and monitoring of the package's location and status throughout its journey from sender to recipient.
client_code	Foreign Key	Serves as a unique identifier for each client or customer who sends or receives packages. It helps link packages to their respective clients, enabling efficient record-keeping and client-specific tracking. This is from the client_information table.
recipient_code	Foreign Key	Provides a unique identifier for each recipient or receiver of packages. It facilitates the association of packages with their intended recipients, aiding in the accurate and timely delivery of items. This is from the recipient_information table.
package_type	-	Classifies packages into different types or categories based on their contents or characteristics. This classification helps handle and route packages appropriately, considering factors such as fragility or special handling requirements.
package_value	-	Indicates the declared or estimated value of each package. It is crucial for assessing insurance coverage and determining the level of care and security required during transport.
collect_amount	-	Represents the amount of money to be collected upon delivery of the package. It is applicable when the recipient is expected to pay for the package or any associated fees upon receipt.
weight	-	Records the weight of each package, typically measured in a standardized unit such as kilograms or pounds.

Table 5. List of Features in Recipient Information in OLTP Table.

Column Name	Key Type	Description
recipient_code	Primary Key	Contains a unique identifier assigned to each recipient or receiver of packages. It is a reference code for precisely identifying recipients and facilitates accurate package delivery.
delivery_island	-	Categorizes the geographical island or region where the delivery is intended. Divided into five: NCR, North Luzon, South Luzon, Visayas, and Mindanao
province	-	Records the specific province or administrative region where the recipient is located.
city	-	Specifies the recipient's city or urban locality. It narrows down the delivery destination within a province or region and is crucial for precise routing.
barangay	-	Identifies the recipient's barangay or neighborhood within a city or municipality. It provides fine-grained location details, aiding in the accuracy of package deliveries.
longitude	-	Stores the geographic longitude coordinate associated with the recipient's location.
latitude	-	Contains the geographic latitude coordinate corresponding to the recipient's location.

Online Analytical Processing (OLAP)

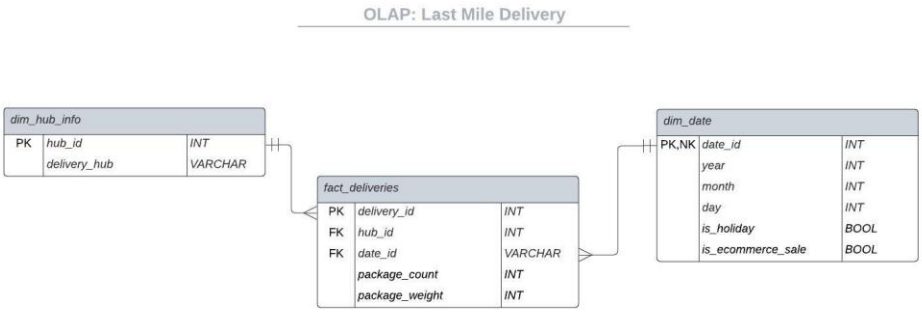


Figure 5. OLAP for Package Delivery Database.

Note: This section only contains an in-depth description, for the full implementation, please see the appendix.

The OLAP system for the Package Delivery Database comprises three key tables: "dim_hub_info," "fact_deliveries," and "dim_date." It is motivated by the need to enable in-depth analytics and data-driven decision-making within the package delivery industry. These tables together facilitate multidimensional analysis, allowing stakeholders to assess hub performance, track delivery trends, and optimize processes, ultimately enhancing operational efficiency and customer experiences in the package delivery sector.

Table 6. List of Features in Fact Deliveries in OLAP Table.

Column Name	Key Type	Description
delivery_id	Primary Key	Serves as a unique identifier for each individual delivery within the system. It distinguishes one delivery from another and facilitates precise tracking and management of delivery records.
hub_id	Foreign Key	Contains an identifier that corresponds to the specific delivery hub or distribution center associated with each delivery. It helps in routing and organizing deliveries efficiently, considering their origination points. This is from the dim_hub_info table.
date_id	Foreign Key	Stores a unique identifier representing specific dates. It is used to associate each delivery with a particular date, allowing for time-based analysis and reporting. This also includes identifiers if the date is a holiday and/or an e-commerce sale date. This is from the dim_date.
package_count	-	Records the number of packages or shipments included in each delivery. It provides information about the volume of transported items within a single delivery.
package_weight	-	Captures the total weight of packages within each delivery. It is typically measured in a standardized unit, such as kilograms or pounds, and helps assess the load and handling requirements for deliveries.

Table 7 List of Features in Dim Hub Info in OLAP Table.

Column Name	Key Type	Description
hub_id	Primary Key	Contains an identifier that corresponds to the specific delivery hub or distribution center associated with each delivery. It helps in routing and organizing deliveries efficiently, considering their origination points.
delivery_hub	-	Provides a descriptive name or label for each delivery hub or distribution center.

Table 8. List of Features in Dim Date in OLAP Table.

Column Name	Key Type	Description
date_id	Primary Key, Natural Key	Serves as a unique identifier for specific dates within the system.
year	-	Indicates the calendar year associated with a particular date. It provides information about the year in which the date falls and is useful for annual trends and comparisons.
month	-	Records the specific month of a given date. It categorizes dates into months, enabling monthly analysis, and is valuable for tracking seasonal trends.
day	-	Represents the day of the month for a given date.
is_holiday	-	Is a binary indicator that denotes whether a specific date is a holiday (1) or not (0). It helps identify holidays for various purposes, such as adjusting delivery schedules or assessing holiday-related impacts.
is_ecommerce_sale		Is a binary indicator that signifies whether a date corresponds to an e-commerce sale event (1) or not (0). It assists in tracking e-commerce sales periods and their potential influence on delivery volumes and trends.

NoSQL (Key-Value Store)

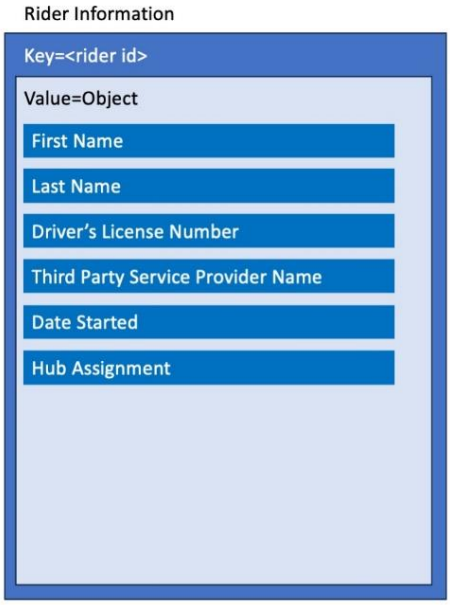


Figure 6. NoSQL for Rider Database.

Note: This section only contains an in-depth description, for the full implementation, please see the appendix.

In storing the rider information, a NoSQL implementation was used. In AWS, this is done through the product of AWS DynamoDB. The need to do a NoSQL database for rider information arises from the fact that the information currently recorded is from the information gathered during package delivery. No other information is gathered, given the nature that TPSP provides the riders. In the future, when rider information is gathered more completely and accurately, the columns can be added with other information such as years of experience, motor vehicle type, and personal historical accidents, hence the space in the. Given that this database can be used to track specific riders, this will be preprocessed to remove this PII, and the anonymized data will then be used for AWS DynamoDB. The following table describes the features of this key-value store.

Table 9. List of Features in Rider Table in DynamoDB.

Column Name	Description
rider_id	Serves as a unique identifier for each rider within the system. It distinguishes one rider from another and facilitates precise tracking and management of rider records.
first_name	Contains the first name of the rider.
last_name	Contains the last name of the rider.
drivers_license_number	The license number of the rider
tpsp_name	Riders may come from different third-party agency. This column contains information which TPSP the rider came from
date_started	The date when the rider started as a rider for the WZP WZP company
hub_assignment	The current hub where the rider delivers.

Extract-Transform-Load (ETL) Jobs

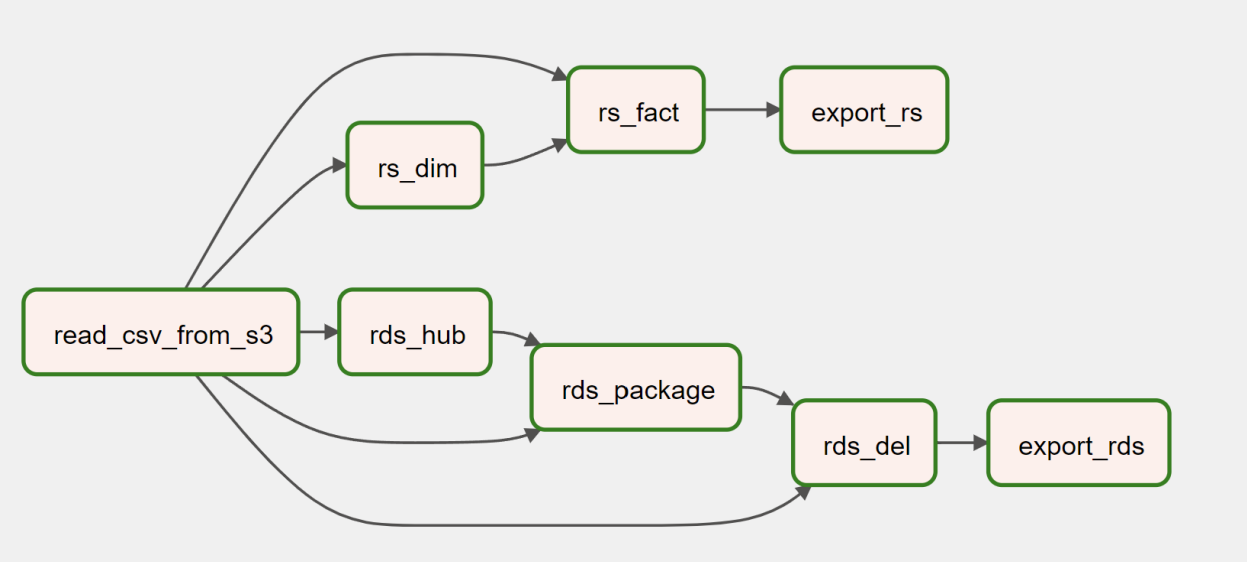


Figure 7. DAG Graph for OLTP and OLAP ETL.

Note: This section only contains an in-depth description, for the full implementation, please see the appendix.

In creating the pipeline of the data architecture, Apache Airflow was used for the ETL work. The pipeline is designed to work in 2 parts: One for the package datasets (for both OLTP and OLAP databases) and the other for the rider data (NoSQL). The process for the package data starts with the user uploading the file into the landing directory of the S3 bucket. The Directed Acyclic Graph (DAG) reads the uploaded package raw data from the S3 bucket and converts it into a pandas dataframe. The DAG's initial output produces a CSV file, which undergoes a process to eliminate duplicate rows before being uploaded to the working directory within the S3 bucket. Users can readily access and utilize this data for their specific tasks and requirements.

For the OLTP, each table column is extracted from the raw data and transformed into corresponding OLTP tables. To optimize efficiency, tables without dependencies are processed first. For instance, tables like *hub, client,* and *recipient* information are prioritized in this initial processing phase. The tables with foreign keys (i.e., *package* and *delivery*) from the other table are processed after the other tables are completed. Lastly, all tables are converted into a CSV file and uploaded to the gold directory of the S3 bucket.

The OLAP pipeline starts with the same dataframe and processes the dimension tables (i.e., *dim_hub_info* and *dim_date*) by selecting data from the raw dataset. Following the completion of dimension tables, the fact table undergoes processing. This involves grouping the dataframe based on identifiers from the dimension tables, such as delivery hub and dispatched date IDs. The objective is to compute aggregates like the total count of packages and their combined weight per hub and date.

Lastly, all tables are converted into a CSV file and uploaded to the gold directory.

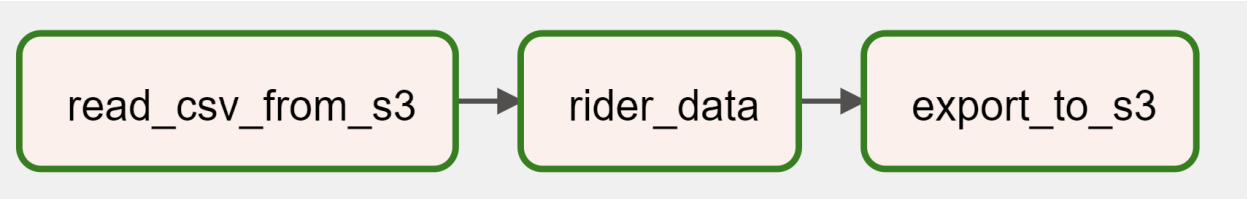


Figure 8. DAG Graph for NoSQL ETL.

Note: This section only contains an in-depth description, for the full implementation, please see the appendix.

For the rider information dataset, the raw data is read from the landing directory of the S3 bucket. The file is transformed into pandas dataframe and then processed by two DynamoDB tables, *rider_data* and *rider_data2*. The *rider_data*, which includes personal information about the riders, is converted into a JSON and stored in the sensitive directory of the S3 bucket to ensure data security. The *rider_data2*, which references a rider via its *rider_id*, is converted into a JSON file and then uploaded in the gold directory making, it easily accessible for users to incorporate into their tasks and analyses.

Both DAGs operate on a daily schedule to align with the daily influx of packages. This ensures that the package dataset remains up-to-date with the real-time delivery operations. Regarding the rider data, since it's sourced from a third-party service with uncertain rider assignments, a daily update is practical. However, if rider assignments are stable, the DAG runtime can be adjusted to match when new riders are expected to join the company, optimizing resource allocation.

Conclusion

The study designed a data architecture for a logistic company with preprocessed data for descriptive and predictive analytics. The raw data is extracted-transformed-loaded to different directories in AWS S3 buckets using different AWS products based on their final use. These transformations are scheduled and managed using Apache Airflow. Raw data is transformed into an OLTP database using AWS RDS, an OLAP database using AWS Redshift, and a key-value store using AWS DynamoDB. The processed data is then loaded into different directories of the AWS S3 bucket, serving as different zones for the data lake. Specific users can only access each zone (read/write). Data Scientists, engineers, stewards, and business analysts have different access levels to the different zones, ensuring proper data governance. The data architecture design works from end-to-end raw data to descriptive/predictive analytics.

Recommendation

The authors recommend extending the architecture to a much wider operation of the logistics company, which can include internal business data storage, finance, human resources, and others. Customer delivery data are not limited to improving the operations, but it can also help analyze its effect on other parts of the business.

Expanding the data architecture consideration beyond data storage is also recommended for Data Engineering work. Operationalization of ML models and ensuring solution quality must also be part of design considerations to ensure the proper use and flow of Data Science solutions.

References

[1] Mordor Intelligence (2022) Philippines CEP Market Size & Share Analysis – Growth Trends & Forecasts (2023 – 2028). Retrieved September 17, 2023, from https://www.mordorintelligence.com/industry-reports/philippines-cep-market

Appendix

Appendix A. Raw Data Sample

Package Dataset

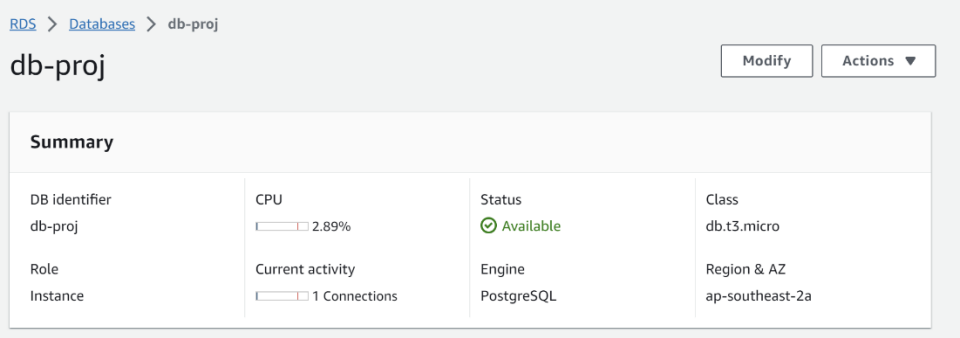
tracking_nı	delivery_h	status	dispatch_date	delivery_date	client_code	client_nam	client_subg	client_subg	payment_n	client_indu	package_ty	package_vı	collect_am	weight	delivery_isl	province
1	MNL	SUCCESS	09/10/2023	09/10/2023	18	wzp_store	18-Jan	wzp_groce	cod	ecomm	small	123	200	1	NCR	NCR
1	MNL	SUCCESS	09/10/2023	09/10/2023	18	wzp_store	18-Jan	wzp_groce	cod	ecomm	small	123	200	1	NCR	NCR
2	MNL	SUCCESS	09/10/2023	09/11/2023	18	wzp_store	18-Jan	wzp_groce	cod	ecomm	medium	987	300	5	NCR	NCR
3	MNL	FAIL	09/10/2023	09/11/2023	18	wzp_store	18-Jan	wzp_groce	cod	ecomm	medium	984	69	5	NCR	NCR
4	CAV	SUCCESS	09/11/2023	09/11/2023	18	wzp_store	18-Feb	wzp_dress	non-cod	ecomm	small	123	0	1	SLUZ	Cavite
5	CAV	SUCCESS	09/10/2023	09/10/2023	18	wzp_store	18-Feb	wzp_dress	cod	ecomm	large	7887	78	7	SLUZ	Cavite
6	BUL	SUCCESS	09/11/2023	09/11/2023	16	wzp_bank	16-Jan	wzp_debit	non-cod	bank	small	100	0	1	NLUZ	Bulacan
7	BUL	FAIL	09/10/2023	09/10/2023	16	wzp_bank	16-Jan	wzp_debit	non-cod	bank	small	1000	0	1	NLUZ	Bulacan
8	CAV	FAIL	09/10/2023	09/11/2023	16	wzp_bank	16-Jan	wzp_debit	non-cod	bank	small	1000	0	1	SLUZ	Cavite
9	MNL	FAIL	09/01/2023	09/12/2023	16	wzp_bank	16-Jan	wzp_debit	non-cod	bank	medium	5000	0	2	NCR	NCR
10	MNL	SUCCESS	09/10/2023	09/10/2023	18	wzp_store	18-Feb	wzp_dress	cod	ecomm	large	65	123	7	NCR	NCR
10	MNL	SUCCESS	09/10/2023	09/10/2023	18	wzp_store	18-Feb	wzp_dress	cod	ecomm	large	65	123	7	NCR	NCR

Rider Dataset

rider_id	first_name	last_name	driver_licer	tpsp_name	date_started	hub_assignment
1	Emilio	Aguinaldo	1001	rider_co	23/08/2021	CAV
2	Manuel	Quezon	1002	rider_co	14/05/2020	MNL
3	Jose	Laurel	1003	rider_co	24/12/2019	MNL
4	Sergio	Osmena	1004	rider_co	22/01/2010	MNL
5	Manuel	Roxas	1005	vroom_vro	24/03/2023	CAV
6	Elpidio	Quirino	1006	vroom_vro	20/06/2022	CAV
7	Ramon	Magsaysay	1007	vroom_vro	01/07/2015	BUL
8	Carlos	Garcia	1008	vroom_vro	07/06/2016	BUL
9	Diosdado	Macapagal	1009	rider_co	11/11/2020	BUL
10	Ferdinand	Marcos	1010	rider_co	12/12/2012	MNL

Appendix B. OLTP (RDS)

Appendix B-1. RDS Details Screenshot



Appendix B-2. SQL Script

RDS (OLTP)

```
%load_ext sql
```

```
%sql postgresql://cpt7:[redacted]@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres
```

```
%%sql

CREATE TABLE hub (
    hub_id SERIAL PRIMARY KEY,
    delivery_hub VARCHAR
);
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'

ResultSet : to convert to pandas, call .DataFrame() or to polars, call .PolarsDataFrame()

```
%%sql

CREATE TABLE client (
    client_id SERIAL PRIMARY KEY,
    client_name VARCHAR,
    client_subgroup_name VARCHAR,
    payment_method VARCHAR,
    client_industry VARCHAR
);
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'

ResultSet : to convert to pandas, call .DataFrame() or to polars, call .PolarsDataFrame()

```
%%sql

CREATE TABLE recipient (
    recipient_code SERIAL PRIMARY KEY,
    delivery_island VARCHAR,
    province VARCHAR,
    city VARCHAR,
    barangay VARCHAR,
    longitude FLOAT,
    latitude FLOAT
);
```

```
%%sql

CREATE TABLE package (
    tracking_number SERIAL PRIMARY KEY,
    client_code INT REFERENCES client,
    recipient_code INT REFERENCES recipient,
    package_type VARCHAR,
    package_value VARCHAR,
    collect_amount INT,
    weight INT
);
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

```
%%sql

DROP TYPE IF EXISTS status_enum;

CREATE TYPE status_enum AS ENUM ('SUCCESS', 'FAIL');

CREATE TABLE delivery (
    delivery_id SERIAL PRIMARY KEY,
    tracking_number INT REFERENCES package,
    hub_id INT REFERENCES hub,
    status status_enum,
    dispatch_date VARCHAR,
    delivery_date VARCHAR
);
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

Appendix B-3. Schema of Tables

```
%%sql

\d
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'

Schema	Name	Type	Owner
public	client	table	cpt7
public	client_client_id_seq	sequence	cpt7
public	delivery	table	cpt7
public	delivery_delivery_id_seq	sequence	cpt7
public	hub	table	cpt7
public	hub_hub_id_seq	sequence	cpt7
public	package	table	cpt7
public	package_tracking_number_seq	sequence	cpt7
public	recipient	table	cpt7
public	recipient_recipient_code_seq	sequence	cpt7

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

Truncated to displaylimit of 10

If you want to see more, please visit displaylimit configuration

```
%%sql

\dt
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'

Schema	Name	Type	Owner
public	client	table	cpt7
public	delivery	table	cpt7
public	hub	table	cpt7
public	package	table	cpt7
public	recipient	table	cpt7

```
postgres=> \d hub
                                  Table "public.hub"
    Column    |       Type        | Collation | Nullable |              Default
--------------+-------------------+-----------+----------+-----------------------------------
 hub_id       | integer           |           | not null | nextval('hub_hub_id_seq'::regclass)
 delivery_hub | character varying |           |          |
Indexes:
    "hub_pkey" PRIMARY KEY, btree (hub_id)
Referenced by:
    TABLE "delivery" CONSTRAINT "delivery_hub_id_fkey" FOREIGN KEY (hub_id) REFERENCES hub(hub_id)

postgres=> \d client
                                     Table "public.client"
       Column        |       Type        | Collation | Nullable |                  Default
---------------------+-------------------+-----------+----------+-------------------------------------------
 client_id           | integer           |           | not null | nextval('client_client_id_seq'::regclass)
 client_name         | character varying |           |          |
 client_subgroup_name | character varying |           |          |
 payment_method      | character varying |           |          |
 client_industry     | character varying |           |          |
Indexes:
    "client_pkey" PRIMARY KEY, btree (client_id)
Referenced by:
    TABLE "package" CONSTRAINT "package_client_code_fkey" FOREIGN KEY (client_code) REFERENCES client(client_id)

postgres=> \d recipient
                                     Table "public.recipient"
     Column      |       Type        | Collation | Nullable |                      Default
-----------------+-------------------+-----------+----------+-----------------------------------------------
 recipient_code  | integer           |           | not null | nextval('recipient_recipient_code_seq'::regclass)
 delivery_island | character varying |           |          |
 province        | character varying |           |          |
 city            | character varying |           |          |
 barangay        | character varying |           |          |
 longitude       | double precision  |           |          |
 latitude        | double precision  |           |          |
Indexes:
    "recipient_pkey" PRIMARY KEY, btree (recipient_code)
Referenced by:
    TABLE "package" CONSTRAINT "package_recipient_code_fkey" FOREIGN KEY (recipient_code) REFERENCES recipient(recipient_code)
postgres=> \d package
                                    Table "public.package"
     Column      |       Type        | Collation | Nullable |                     Default
-----------------+-------------------+-----------+----------+-------------------------------------------------
 tracking_number | integer           |           | not null | nextval('package_tracking_number_seq'::regclass)
 client_code     | integer           |           |          |
 recipient_code  | integer           |           |          |
 package_type    | character varying |           |          |
 package_value   | character varying |           |          |
 collect_amount  | integer           |           |          |
 weight          | integer           |           |          |
Indexes:
    "package_pkey" PRIMARY KEY, btree (tracking_number)
Foreign-key constraints:
    "package_client_code_fkey" FOREIGN KEY (client_code) REFERENCES client(client_id)
    "package_recipient_code_fkey" FOREIGN KEY (recipient_code) REFERENCES recipient(recipient_code)
Referenced by:
    TABLE "delivery" CONSTRAINT "delivery_tracking_number_fkey" FOREIGN KEY (tracking_number) REFERENCES package(tracking_number)

postgres=> \d delivery
                                    Table "public.delivery"
     Column      |       Type        | Collation | Nullable |                    Default
-----------------+-------------------+-----------+----------+-----------------------------------------------
 delivery_id     | integer           |           | not null | nextval('delivery_delivery_id_seq'::regclass)
 tracking_number | integer           |           |          |
 hub_id          | integer           |           |          |
 status          | status_enum       |           |          |
 dispatch_date   | character varying |           |          |
 delivery_date   | character varying |           |          |
Indexes:
    "delivery_pkey" PRIMARY KEY, btree (delivery_id)
Foreign-key constraints:
    "delivery_hub_id_fkey" FOREIGN KEY (hub_id) REFERENCES hub(hub_id)
    "delivery_tracking_number_fkey" FOREIGN KEY (tracking_number) REFERENCES package(tracking_number)
```

Appendix B-4. Sample Contents of Each Table

```
%%sql

SELECT * FROM hub
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'
3 rows affected.

hub_id	delivery_hub
1	MNL
2	CAV
3	BUL

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

```
%%sql

SELECT * FROM client
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'
4 rows affected.

client_id	client_name	client_subgroup_name	payment_method	client_industry
1	wzp_store	wzp_grocery	cod	ecomm
2	wzp_store	wzp_dress	non-cod	ecomm
3	wzp_store	wzp_dress	cod	ecomm
4	wzp_bank	wzp_debit	non-cod	bank

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

```
%%sql

SELECT * FROM recipient
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'
10 rows affected.

recipient_code	delivery_island	province	city	barangay	longitude	latitude
1	NCR	NCR	Manila	Mabait	121.3516	12.3456
2	NCR	NCR	Manila	Masipag	121.6789	12.8429
3	NCR	NCR	Manila	Mabait	121.4563	12.9637
4	SLUZ	Cavite	Cavite City	Maalalahanin	121.5045	12.1295
5	SLUZ	Cavite	Cavite City	Maalalahanin	121.3474	12.0543
6	NLUZ	Bulacan	Malolos	Matahimik	121.5604	12.5639
7	NLUZ	Bulacan	Malolos	Matahimik	121.5697	12.6394
8	SLUZ	Cavite	Cavite City	Maalalahanin	121.2323	12.9583
9	NCR	NCR	Quezon City	Masipag	121.4578	12.9067
10	NCR	NCR	Quezon City	Mabait	121.2323	12.8728

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`
Truncated to displaylimit of 10
If you want to see more, please visit displaylimit configuration

```
%%sql

SELECT * FROM package
```

Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'
10 rows affected.

tracking_number	client_code	recipient_code	package_type	package_value	collect_amount	weight
1	1	1	small	123	200	1
2	1	2	medium	987	300	5
3	1	3	medium	984	69	5
4	2	4	small	123	0	1
5	3	5	large	7887	78	7
6	4	6	small	100	0	1
7	4	7	small	1000	0	1
8	4	8	small	1000	0	1
9	4	9	medium	5000	0	2
10	3	10	large	65	123	7

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`
Truncated to displaylimit of 10
If you want to see more, please visit displaylimit configuration

```
%%sql

SELECT * FROM delivery
```

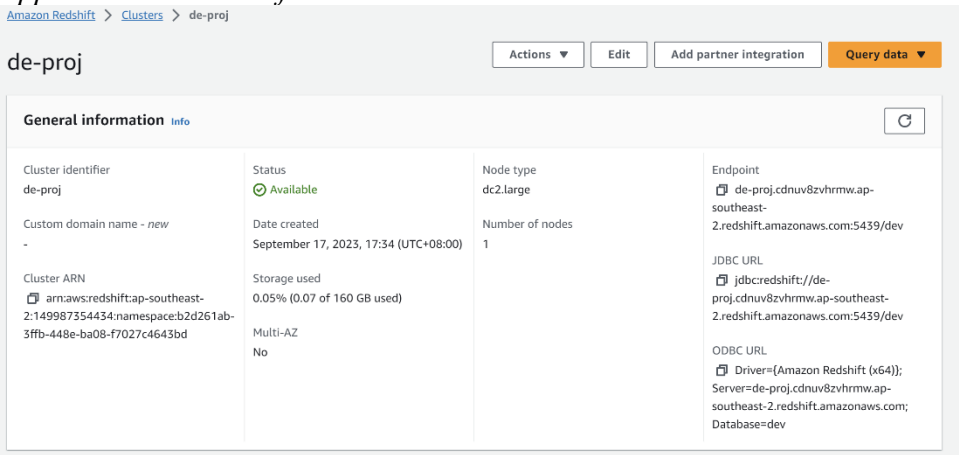
Running query in 'postgresql://cpt7:***@db-proj.csiigtklcjns.ap-southeast-2.rds.amazonaws.com/postgres'
10 rows affected.

delivery_id	tracking_number	hub_id	status	dispatch_date	delivery_date
1	1	1	SUCCESS	9/10/2023	9/10/2023
2	2	1	SUCCESS	9/10/2023	9/11/2023
3	3	1	FAIL	9/10/2023	9/11/2023
4	4	2	SUCCESS	9/11/2023	9/11/2023
5	5	2	SUCCESS	9/10/2023	9/10/2023
6	6	3	SUCCESS	9/11/2023	9/11/2023
7	7	3	FAIL	9/10/2023	9/10/2023
8	8	2	FAIL	9/10/2023	9/11/2023
9	9	1	FAIL	9/1/2023	9/12/2023
10	10	1	SUCCESS	9/10/2023	9/10/2023

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`
Truncated to displaylimit of 10
If you want to see more, please visit displaylimit configuration

Appendix C. OLAP (Redshift)

Appendix C-1. Redshift Details Screenshot



Appendix C-2. SQL Script

Redshift (OLAP)

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%sql redshift://cpt7:[redacted]@de-proj.cdnuv8zvhrmw.ap-southeast-2.redshift.amazonaws.com:5439/dev
```

```
%%sql

CREATE TABLE IF NOT EXISTS dim_hub_info (
    hub_id INTEGER PRIMARY KEY,
    delivery_hub VARCHAR
)

DISTSTYLE ALL;
```

Running query in 'redshift://cpt7:***@de-proj.cdnuv8zvhrmw.ap-southeast-2.redshift.amazonaws.com:5439/dev'

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

```
%%sql

CREATE TABLE IF NOT EXISTS dim_date (
    date_id INTEGER PRIMARY KEY,
    year INTEGER,
    month INTEGER,
    day INTEGER,
    is_holiday BOOL,
    is_ecommerce_sale BOOL
)

DISTSTYLE ALL;
```

Running query in 'redshift://cpt7:***@de-proj.cdnuv8zvhrmw.ap-southeast-2.redshift.amazonaws.com:5439/dev'

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

```
%%sql

CREATE TABLE IF NOT EXISTS fact_deliveries (
    delivery_id INTEGER PRIMARY KEY,
    hub_id INTEGER DISTKEY REFERENCES dim_hub_info(hub_id),
    date_id VARCHAR REFERENCES dim_date(date_id),
    package_count INTEGER,
    package_weight INTEGER
)
COMPOUND SORTKEY (hub_id, date_id)
```

Running query in 'redshift://cpt7:***@de-proj.cdnuv8zvhrmw.ap-southeast-2.redshift.amazonaws.com:5439/dev'

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

Appendix C-3. Schema of Tables

```
%sql SELECT datname FROM pg_database;
```

Running query in 'redshift://cpt7:***@de-proj.cdnuv8zvhrmw.ap-southeast-2.redshift.amazonaws.com:5439/dev'

5 rows affected.

datname
awsdatacatalog
dev
padb_harvest
template1
template0

ResultSet : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

```
%%sql
\d
```

Running query in 'redshift://cpt7:***@de-proj.cdnuv8zvhrmw.ap-southeast-2.redshift.amazonaws.com:5439/dev'

schema	name	type	owner
public	dim_date	table	cpt7
public	dim_hub_info	table	cpt7
public	fact_deliveries	table	cpt7

ResultSet : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

```
dev=# \d dim_hub_info
                    Table "public.dim_hub_info"
    Column    |          Type          | Collation | Nullable | Default
--------------+------------------------+-----------+----------+---------
 hub_id       | integer                |           | not null |
 delivery_hub | character varying(256) |           |          |
Indexes:
    "dim_hub_info_pkey" PRIMARY KEY, btree (hub_id)
Referenced by:
    TABLE "fact_deliveries" CONSTRAINT "fact_deliveries_hub_id_fkey" FOREIGN KEY (hub_id) REFERENCES dim_hub_info(hub_id)

dev=# \d dim_date
                  Table "public.dim_date"
      Column       |  Type   | Collation | Nullable | Default
-------------------+---------+-----------+----------+---------
 date_id           | integer |           | not null |
 year              | integer |           |          |
 month             | integer |           |          |
 day               | integer |           |          |
 is_holiday        | boolean |           |          |
 is_ecommerce_sale | boolean |           |          |
Indexes:
    "dim_date_pkey" PRIMARY KEY, btree (date_id)
Referenced by:
    TABLE "fact_deliveries" CONSTRAINT "fact_deliveries_date_id_fkey" FOREIGN KEY (date_id) REFERENCES dim_date(date_id)

dev=# \d fact_deliveries
                     Table "public.fact_deliveries"
     Column     |          Type          | Collation | Nullable | Default
----------------+------------------------+-----------+----------+---------
 delivery_id    | integer                |           | not null |
 hub_id         | integer                |           |          |
 date_id        | character varying(256) |           |          |
 package_count  | integer                |           |          |
 package_weight | integer                |           |          |
Indexes:
    "fact_deliveries_pkey" PRIMARY KEY, btree (delivery_id)
Foreign-key constraints:
    "fact_deliveries_date_id_fkey" FOREIGN KEY (date_id) REFERENCES dim_date(date_id)
    "fact_deliveries_hub_id_fkey" FOREIGN KEY (hub_id) REFERENCES dim_hub_info(hub_id)
```

Appendix C-4. Sample Contents of Each Table

```
%%sql

SELECT * FROM dim_hub_info
```

Running query in 'redshift://cpt7:***@de-proj.cdnuv8zvhrmw.ap-southeast-2.redshift.amazonaws.com:5439/dev'
3 rows affected.

hub_id	delivery_hub
1	MNL
2	CAV
3	BUL

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

```
%%sql

SELECT * FROM dim_date
```

Running query in 'redshift://cpt7:***@de-proj.cdnuv8zvhrmw.ap-southeast-2.redshift.amazonaws.com:5439/dev'
3 rows affected.

date_id	year	month	day	is_holiday	is_ecommerce_sale
9102023	2023	9	10	False	False
9112023	2023	9	11	False	True
912023	2023	9	1	True	False

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

```
%%sql

SELECT * FROM fact_deliveries
```

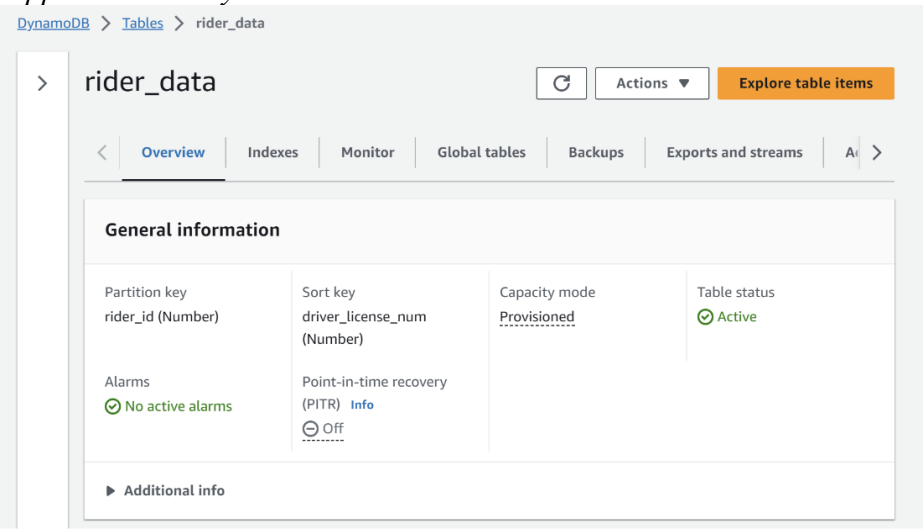
Running query in 'redshift://cpt7:***@de-proj.cdnuv8zvhrmw.ap-southeast-2.redshift.amazonaws.com:5439/dev'
6 rows affected.

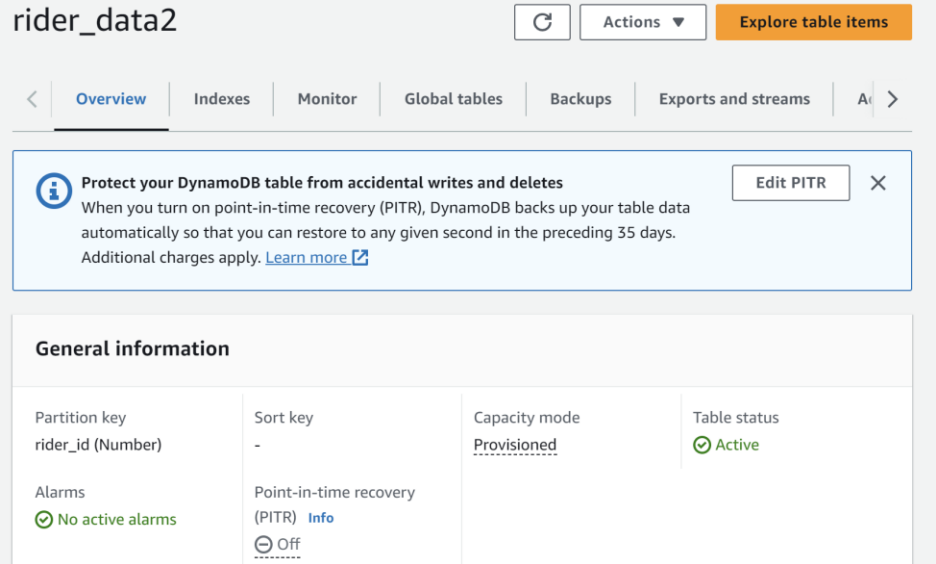
delivery_id	hub_id	date_id	package_count	package_weight
2	2	9102023	2	8
3	2	9112023	1	1
0	3	9102023	1	1
1	3	9112023	1	1
4	1	912023	1	2
5	1	9102023	6	26

`ResultSet` : to convert to pandas, call `.DataFrame()` or to polars, call `.PolarsDataFrame()`

Appendix C. NoSQL (DynamoDB)

Appendix C-1. DynamoDB Details Screenshot





Appendix C-2. Sample Contents of DynamoDB

DynamoDB (NoSQL)

```
import boto3
```

```
dynamodb = boto3.resource('dynamodb')
rider = dynamodb.Table('rider_data')
client = boto3.client('dynamodb')
```

```
client.execute_statement(
    Statement = """
    SELECT * FROM rider_data
    """
)['Items']
```

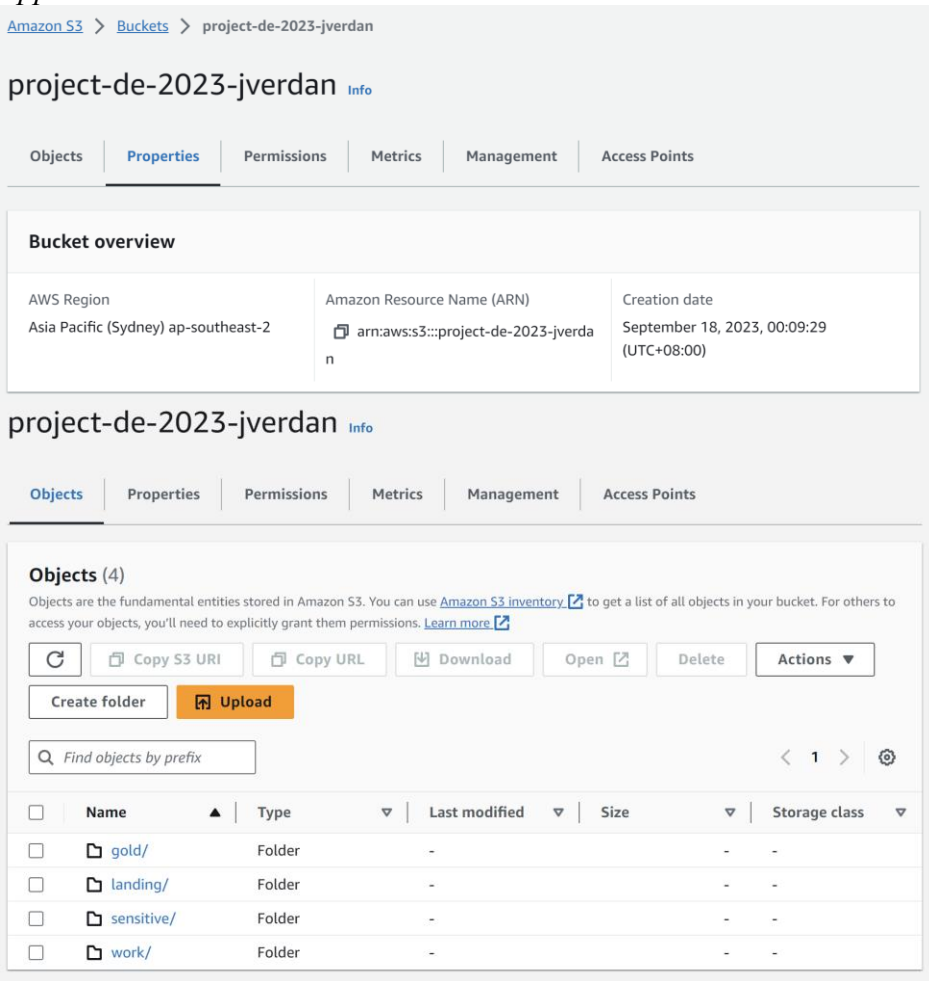
```
[{'rider_id': {'N': '7'},
  'date_started': {'S': '2015-07-01'},
  'last_name': {'S': 'Magsaysay'},
  'driver_license_num': {'N': '1007'},
  'first_name': {'S': 'Ramon'},
  'tpsp_name': {'S': 'vroom_vroom'},
  'hub_assignment': {'S': 'BUL'}},
 {'rider_id': {'N': '8'},
  'date_started': {'S': '2016-06-07'},
  'last_name': {'S': 'Garcia'},
  'driver_license_num': {'N': '1008'},
  'first_name': {'S': 'Carlos'},
  'tpsp_name': {'S': 'vroom_vroom'},
  'hub_assignment': {'S': 'BUL'}},
 {'rider_id': {'N': '10'},
  'date_started': {'S': '2012-12-12'},
  'last_name': {'S': 'Marcos'},
  'driver_license_num': {'N': '1010'},
  'first_name': {'S': 'Ferdinand'},
  'tpsp_name': {'S': 'rider_co'},
  'hub_assignment': {'S': 'MNL'}},
 {'rider_id': {'N': '3'},
  'date_started': {'S': '2019-12-24'},
  'last_name': {'S': 'Laurel'},
  'driver_license_num': {'N': '1003'},
  'first_name': {'S': 'Jose'},
  'tpsp_name': {'S': 'rider_co'},
  'hub_assignment': {'S': 'MNL'}},
 {'rider_id': {'N': '2'},
  'date_started': {'S': '2020-05-14'},
  'last_name': {'S': 'Quezon'},
  'driver_license_num': {'N': '1002'},
  'first_name': {'S': 'Manuel'},
  'tpsp_name': {'S': 'rider_co'},
  'hub_assignment': {'S': 'MNL'}},
 {'rider_id': {'N': '9'},
  'date_started': {'S': '2020-11-11'},
  'last_name': {'S': 'Macapagal'},
  'driver_license_num': {'N': '1009'},
  'first_name': {'S': 'Diosdado'},
  'tpsp_name': {'S': 'rider_co'},
  'hub_assignment': {'S': 'BUL'}},
 {'rider_id': {'N': '4'},
  'date_started': {'S': '2010-01-22'},
  'last_name': {'S': 'Osmena'},
  'driver_license_num': {'N': '1004'},
  'first_name': {'S': 'Sergio'},
  'tpsp_name': {'S': 'rider_co'},
  'hub_assignment': {'S': 'MNL'}},
 {'rider_id': {'N': '6'},
  'date_started': {'S': '2022-06-20'},
  'last_name': {'S': 'Quirino'},
  'driver_license_num': {'N': '1006'},
  'first_name': {'S': 'Elpidio'},
  'tpsp_name': {'S': 'vroom_vroom'},
  'hub_assignment': {'S': 'CAV'}},
 {'rider_id': {'N': '1'},
  'date_started': {'S': '2021-08-23'},
  'last_name': {'S': 'Aguinaldo'},
  'driver_license_num': {'N': '1001'},
  'first_name': {'S': 'Emilio'},
  'tpsp_name': {'S': 'rider_co'},
  'hub_assignment': {'S': 'CAV'}},
```

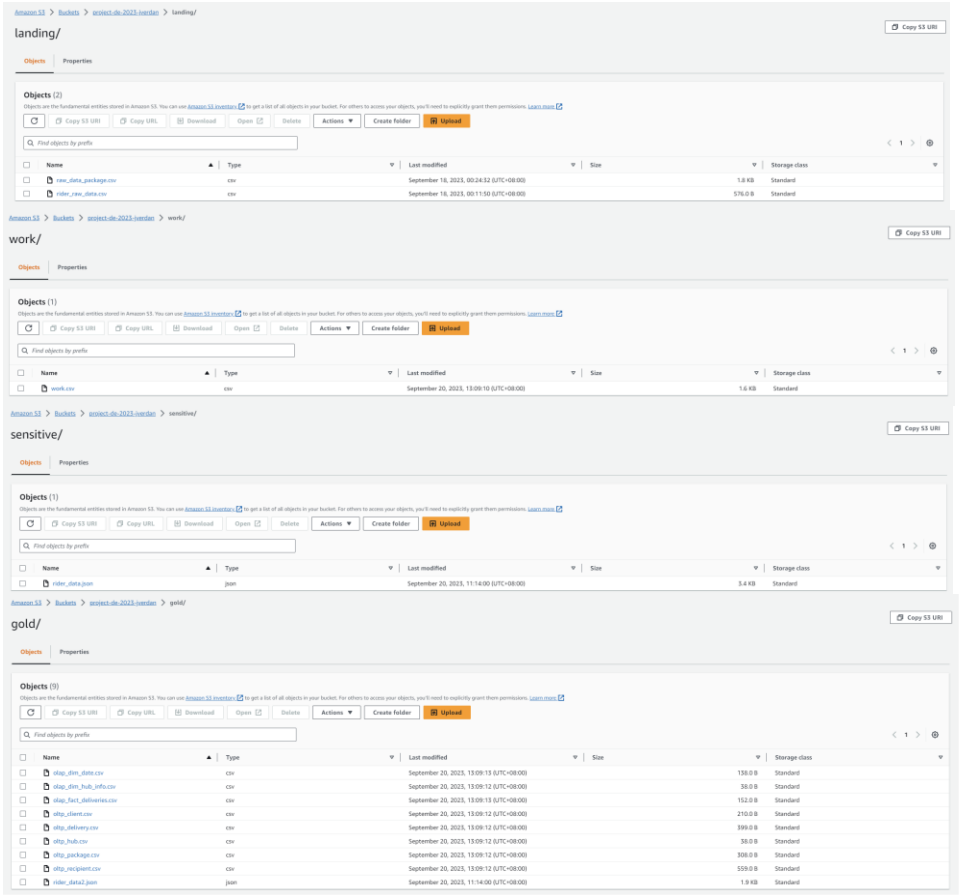
```
client.execute_statement(
    Statement = """
    SELECT * FROM rider_data2
    """
)['Items']
```

```
[{'rider_id': {'N': '7'},
  'tpsp_name': {'S': 'vroom_vroom'},
  'date_started': {'S': '2015-07-01'},
  'hub_assignment': {'S': 'BUL'}},
 {'rider_id': {'N': '8'},
  'tpsp_name': {'S': 'vroom_vroom'},
  'date_started': {'S': '2016-06-07'},
  'hub_assignment': {'S': 'BUL'}},
 {'rider_id': {'N': '10'},
  'tpsp_name': {'S': 'rider_co'},
  'date_started': {'S': '2012-12-12'},
  'hub_assignment': {'S': 'MNL'}},
 {'rider_id': {'N': '3'},
  'tpsp_name': {'S': 'rider_co'},
  'date_started': {'S': '2019-12-24'},
  'hub_assignment': {'S': 'MNL'}},
 {'rider_id': {'N': '2'},
  'tpsp_name': {'S': 'rider_co'},
  'date_started': {'S': '2020-05-14'},
  'hub_assignment': {'S': 'MNL'}},
 {'rider_id': {'N': '9'},
  'tpsp_name': {'S': 'rider_co'},
  'date_started': {'S': '2020-11-11'},
  'hub_assignment': {'S': 'BUL'}},
 {'rider_id': {'N': '4'},
  'tpsp_name': {'S': 'rider_co'},
  'date_started': {'S': '2010-01-22'},
  'hub_assignment': {'S': 'MNL'}},
 {'rider_id': {'N': '6'},
  'tpsp_name': {'S': 'vroom_vroom'},
  'date_started': {'S': '2022-06-20'},
  'hub_assignment': {'S': 'CAV'}},
 {'rider_id': {'N': '1'},
  'tpsp_name': {'S': 'rider_co'},
  'date_started': {'S': '2021-08-23'},
  'hub_assignment': {'S': 'CAV'}},
 {'rider_id': {'N': '5'},
  'tpsp_name': {'S': 'vroom_vroom'},
  'date_started': {'S': '2023-03-24'},
  'hub_assignment': {'S': 'CAV'}}]
```

Appendix D. Data Lake (S3 Buckets)

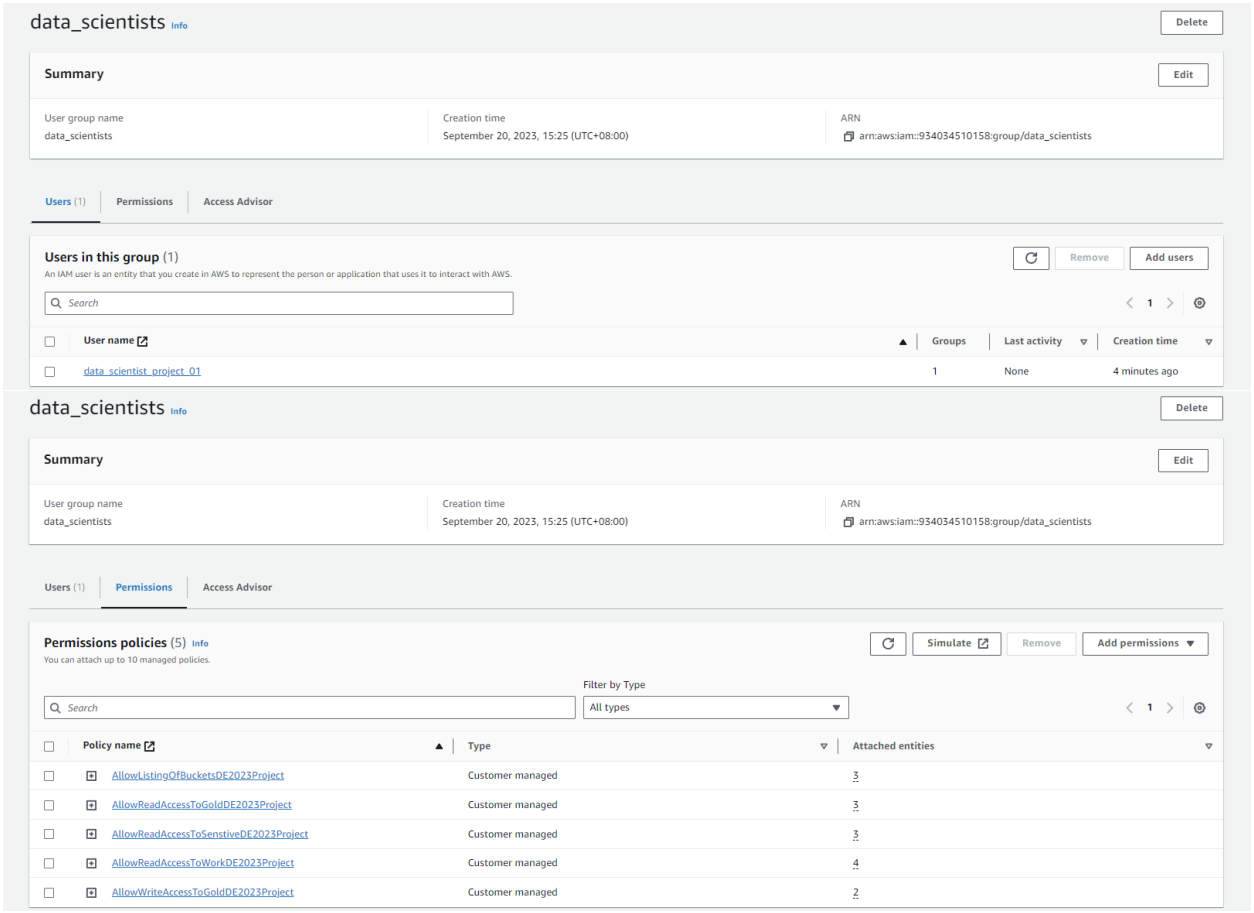
Appendix D-1. S3 Bucket and its Directories Screenshot



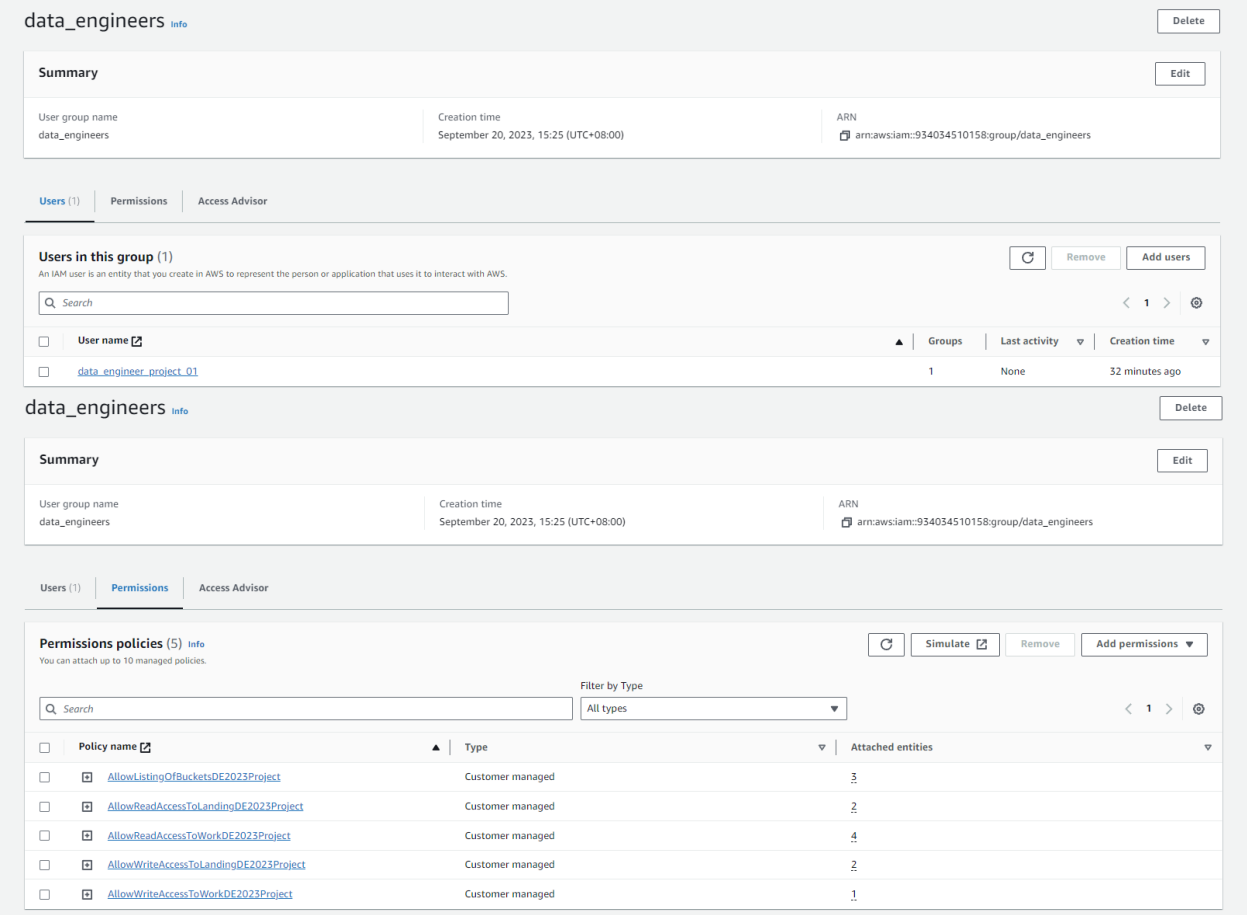

Amazon S3 > Buckets > project-de-2023-iverdan > landing/
landing/
Copy S3 URI
Objects
Properties
Objects (2)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
Name
Type
Last modified
Size
Storage class
raw_data_package.csv
csv
September 18, 2023, 00:24:32 (UTC+08:00)
1.8 KB
Standard
rider_raw_data.csv
csv
September 18, 2023, 00:11:50 (UTC+08:00)
576.0 B
Standard
Amazon S3 > Buckets > project-de-2023-iverdan > work/
work/
Copy S3 URI
Objects
Properties
Objects (1)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
Name
Type
Last modified
Size
Storage class
work.csv
csv
September 20, 2023, 13:09:10 (UTC+08:00)
1.6 KB
Standard
Amazon S3 > Buckets > project-de-2023-iverdan > sensitive/
sensitive/
Copy S3 URI
Objects
Properties
Objects (1)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
Name
Type
Last modified
Size
Storage class
rider_data.json
json
September 20, 2023, 11:14:00 (UTC+08:00)
3.4 KB
Standard
Amazon S3 > Buckets > project-de-2023-iverdan > gold/
gold/
Copy S3 URI
Objects
Properties
Objects (9)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
Name
Type
Last modified
Size
Storage class
olap_dim_date.csv
csv
September 20, 2023, 13:09:13 (UTC+08:00)
138.0 B
Standard
olap_dim_hub_info.csv
csv
September 20, 2023, 13:09:12 (UTC+08:00)
38.0 B
Standard
olap_fact_deliveries.csv
csv
September 20, 2023, 13:09:13 (UTC+08:00)
152.0 B
Standard
oltp_client.csv
csv
September 20, 2023, 13:09:12 (UTC+08:00)
210.0 B
Standard
oltp_delivery.csv
csv
September 20, 2023, 13:09:12 (UTC+08:00)
399.0 B
Standard
oltp_hub.csv
csv
September 20, 2023, 13:09:12 (UTC+08:00)
38.0 B
Standard
oltp_package.csv
csv
September 20, 2023, 13:09:12 (UTC+08:00)
308.0 B
Standard
oltp_recipient.csv
csv
September 20, 2023, 13:09:12 (UTC+08:00)
559.0 B
Standard
rider_data2.json
json
September 20, 2023, 11:14:00 (UTC+08:00)
1.9 KB
Standard

Appendix D-2. IAM User and Roles Permissions

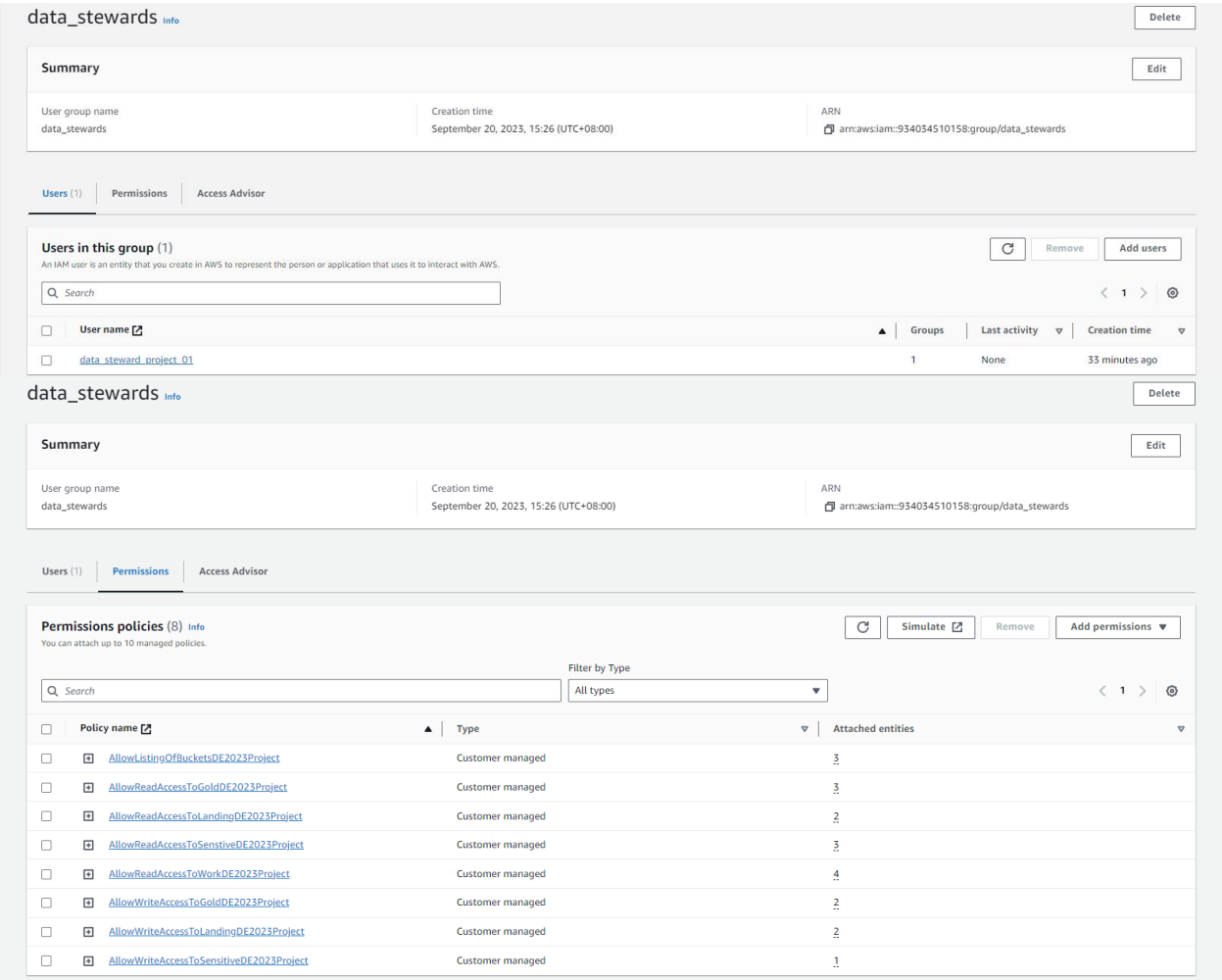
Data Scientist



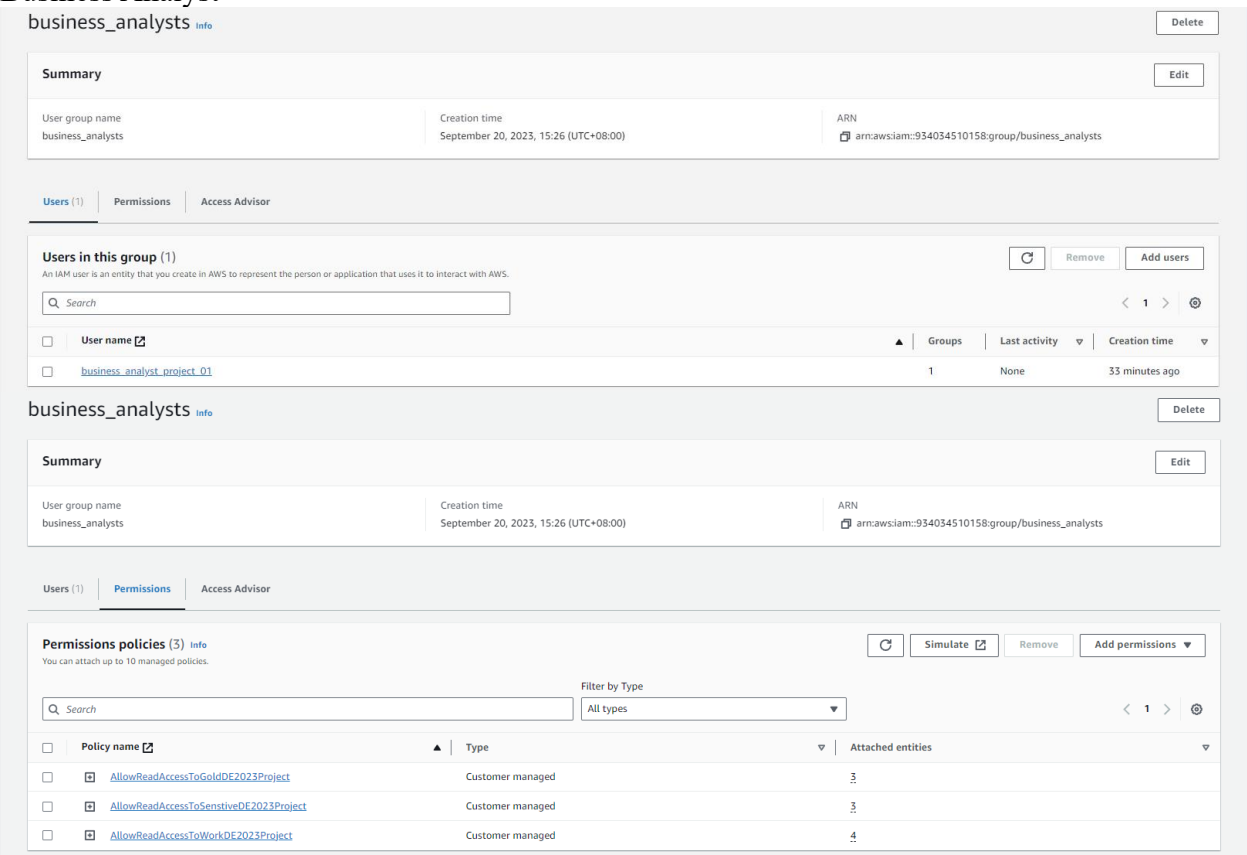
Data Engineers



Data Stewards

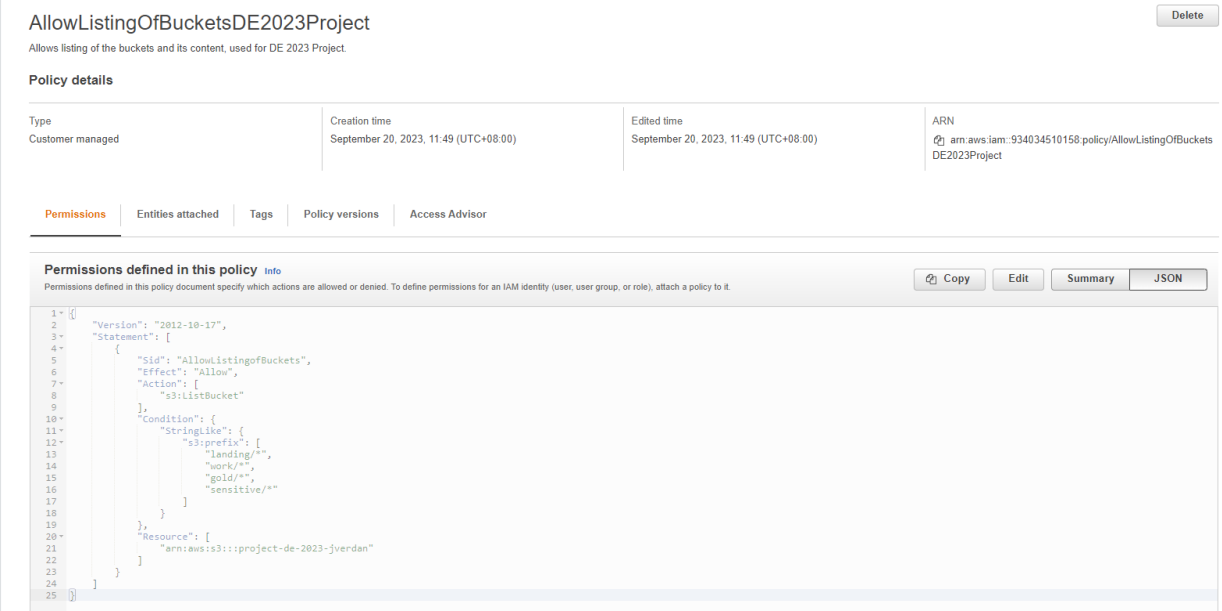


Business Analyst



Appendix D-3. Permissions JSON Code

Listing of Buckets



Read Access

AllowReadAccessToLandingDE2023Project

Delete

Allows read access to the landing directory and its content, used for DE 2023 Project.

Policy details

Type	Creation time	Edited time	ARN
Customer managed	September 20, 2023, 11:58 (UTC+08:00)	September 20, 2023, 11:58 (UTC+08:00)	arn:aws:iam::934034510158:policy/AllowReadAccessToLandingDE2023Project

Permissions | Entities attached | Tags | Policy versions | Access Advisor

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Copy | Edit | Summary | JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReadAccessToLanding",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::project-de-2023-jverdan/landing/*"
        }
    ]
}
```

AllowReadAccessToWorkDE2023Project

Delete

Allows read access to the work directory and its content, used for DE 2023 Project.

Policy details

Type	Creation time	Edited time	ARN
Customer managed	September 20, 2023, 13:13 (UTC+08:00)	September 20, 2023, 13:13 (UTC+08:00)	arn:aws:iam::934034510158:policy/AllowReadAccessToWorkDE2023Project

Permissions | Entities attached | Tags | Policy versions | Access Advisor

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Copy | Edit | Summary | JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReadAccessToWork",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::project-de-2023-jverdan/work/*"
        }
    ]
}
```

AllowReadAccessToGoldDE2023Project

Delete

Allows read access to the gold directory and its content, used for DE 2023 Project.

Policy details

Type	Creation time	Edited time	ARN
Customer managed	September 20, 2023, 13:17 (UTC+08:00)	September 20, 2023, 13:17 (UTC+08:00)	arn:aws:iam::934034510158:policy/AllowReadAccessToGoldDE2023Project

Permissions | Entities attached | Tags | Policy versions | Access Advisor

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Copy | Edit | Summary | JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReadAccessToGold",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::project-de-2023-jverdan/gold/*"
        }
    ]
}
```

AllowReadAccessToSenstiveDE2023Project

Delete

Allows read access to the sensitive directory and its content, used for DE 2023 Project.

Policy details

Type	Creation time	Edited time	ARN
Customer managed	September 20, 2023, 13:19 (UTC+08:00)	September 20, 2023, 13:19 (UTC+08:00)	arn:aws:iam::934034510158:policy/AllowReadAccessToSenstiveDE2023Project

Permissions | Entities attached | Tags | Policy versions | Access Advisor

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Copy | Edit | Summary | JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReadAccessToSensitive",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::project-de-2023-jverdan/sensitive/*"
        }
    ]
}
```

Write Access

AllowWriteAccessToLandingDE2023Project

Delete

Allows write access to the landing directory and its content, used for DE 2023 Project.

Policy details

Type	Creation time	Edited time	ARN
Customer managed	September 20, 2023, 13:12 (UTC+08:00)	September 20, 2023, 13:12 (UTC+08:00)	arn:aws:iam::934034510158:policy/AllowWriteAccessToLandingDE2023Project

Permissions | Entities attached | Tags | Policy versions | Access Advisor

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Copy | Edit | Summary | JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowWriteAccessToLanding",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::project-de-2023-jverdan/landing/*"
        }
    ]
}
```

AllowWriteAccessToWorkDE2023Project

Delete

Allows write access to the work directory and its content, used for DE 2023 Project.

Policy details

Type	Creation time	Edited time	ARN
Customer managed	September 20, 2023, 13:15 (UTC+08:00)	September 20, 2023, 13:15 (UTC+08:00)	arn:aws:iam::934034510158:policy/AllowWriteAccessToWorkDE2023Project

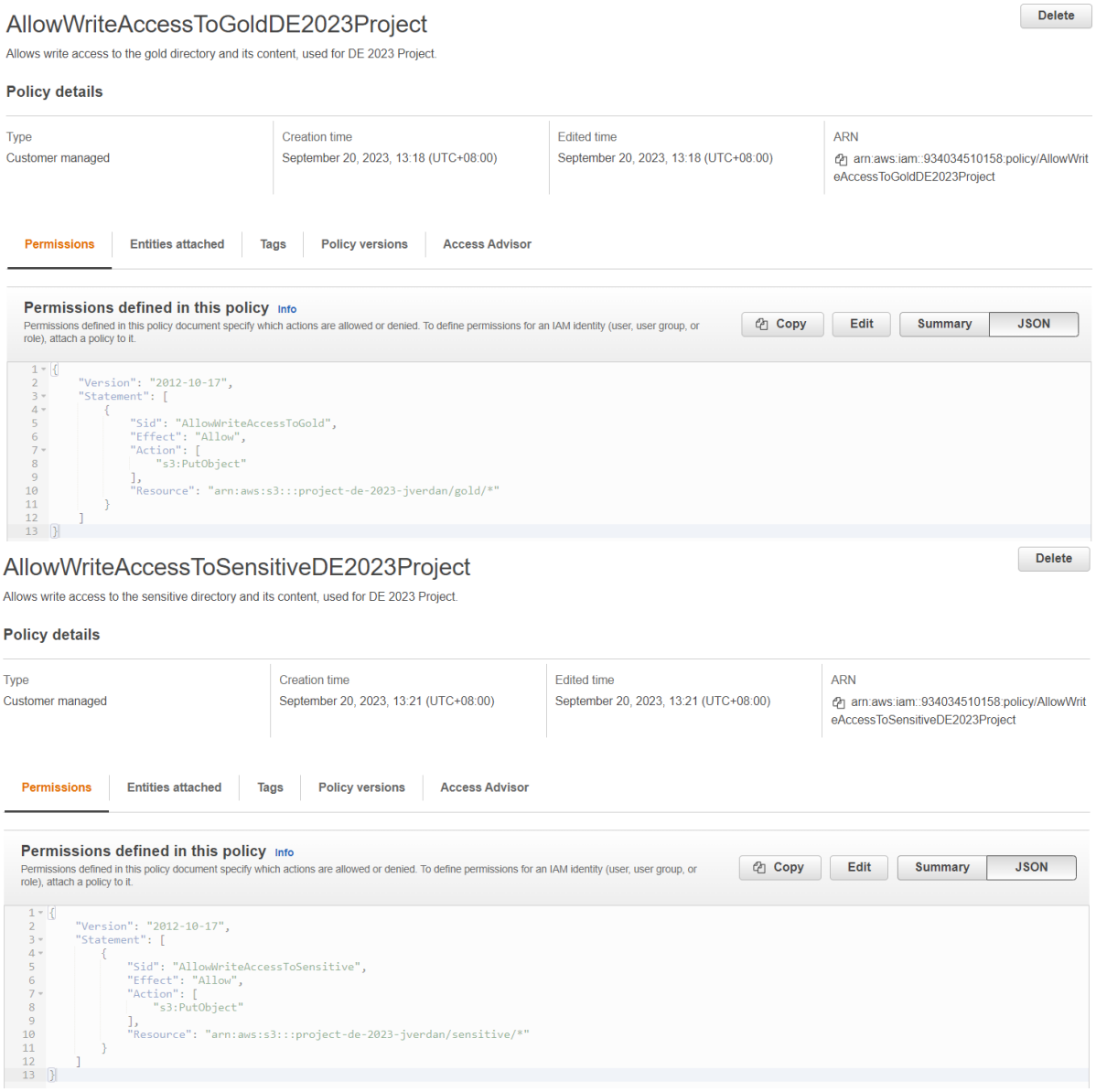
Permissions | Entities attached | Tags | Policy versions | Access Advisor

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

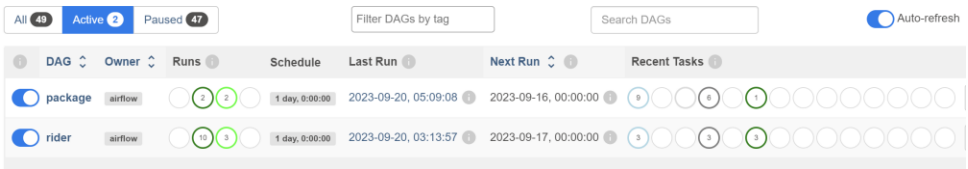
Copy | Edit | Summary | JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowWriteAccessToWork",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::project-de-2023-jverdan/work/*"
        }
    ]
}
```



Appendix E. Workflow Manager (Airflow)

Appendix E-1. Airflow DAGs



Appendix E-2. OLTP and OLAP ETL Code

```
import datetime
from datetime import datetime as dt
import pandas as pd
import boto3
from io import BytesIO
from airflow.decorators import dag, task
from airflow import DAG, Dataset
from airflow.operators.bash import BashOperator
from airflow.hooks.base import BaseHook
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.providers.amazon.aws.hooks.redshift_sql import RedshiftSQLHook
import pyarrow


@dag(
    start_date=datetime.datetime(2023, 9, 15),
    schedule="@daily"
)

def package():
    @task
    def read_csv_from_s3():
        s3_bucket = 'project-de-2023-jverdan'
        s3_key = 'landing/raw_data_package.csv'

        aws_credentials = BaseHook.get_connection('project_iam')

        # Create an S3 client
        s3_client = boto3.client(
        's3',
        aws_access_key_id=aws_credentials.login,
        aws_secret_access_key=aws_credentials.password
    )

        # Use the S3 client to download the file
        response = s3_client.get_object(Bucket=s3_bucket, Key=s3_key)
        csv_content = response['Body'].read()

        # Load CSV data into a Pandas DataFrame
        df_raw = pd.read_csv(BytesIO(csv_content))
        df_work = df_raw.drop_duplicates()

        s3_key2 = f'work/work.csv'
        df_work.to_csv(f'/tmp/work.csv', index=False, encoding='utf-8', sep=',')
        s3_client.upload_file(f'/tmp/work.csv', s3_bucket, s3_key2)

        return df_raw
```

```python
@task
def rds_hub(df):
    hook = PostgresHook(postgres_conn_id='project_rds')
    conn = hook.get_conn()
    cursor = conn.cursor()

    query = '''SELECT hub_id
            FROM hub
            ORDER BY hub_id DESC
            LIMIT 1;'''
    try:
        hub_id = int(cursor.execute(query).fetchone()[0])
    except:
        hub_id = 0

    cursor.execute('SELECT DISTINCT delivery_hub FROM hub')
    hubs = set(row[0] for row in cursor.fetchall())

    client_id = 0
    client = set()
    recipient_id = 0
    recipient = set()

    cursor.execute('DELETE FROM delivery;')
    cursor.execute('DELETE FROM package;')
    cursor.execute('DELETE FROM client;')
    cursor.execute('DELETE FROM recipient;')

    for index, row in df.iterrows():
        if row['delivery_hub'] not in hubs:
            hub_id += 1
            cursor.execute(
                '''INSERT INTO hub (hub_id, delivery_hub)
                VALUES (%s, %s)
                ON CONFLICT (hub_id) DO NOTHING;
                ''', (hub_id, row['delivery_hub'])
            )
            hubs.add(row['delivery_hub'])

     if ((row['client_name'], row['client_subgroup_name'], row['payment_method'])) not in client:
         client_id += 1
         cursor.execute(
             '''INSERT INTO client (client_id, client_name, client_subgroup_name, payment_method, client_industry)
             VALUES (%s, %s, %s, %s, %s)
             ON CONFLICT (client_id) DO NOTHING;
             ''', (client_id, row['client_name'], row['client_subgroup_name'], row['payment_method'], row['client_industry'])
         )
         client.add((row['client_name'], row['client_subgroup_name'], row['payment_method']))

     if (row['longitude'], row['latitude']) not in recipient:
         recipient_id += 1
         cursor.execute(
             '''INSERT INTO recipient (recipient_code, delivery_island, province, city, barangay, longitude, latitude)
             VALUES (%s, %s, %s, %s, %s, %s, %s)
             ON CONFLICT (recipient_code) DO NOTHING;
             ''', (recipient_id, row['delivery_island'], row['province'], row['city'],
                   row['barangay'], row['longitude'], row['latitude'])
         )
         recipient.add((row['longitude'], row['latitude']))

  conn.commit()
  cursor.close()

@task
def rds_package(df):
    hook = PostgresHook(postgres_conn_id='project_rds')
    conn = hook.get_conn()
    cursor = conn.cursor()

    tracking = set()
    for index, row in df.iterrows():
        query = "SELECT client_id FROM client WHERE client_name = %s AND client_subgroup_name = %s AND payment_method = %s"
        cursor.execute(query, (row['client_name'], row['client_subgroup_name'], row['payment_method']))
        client_id = cursor.fetchone()[0]
        query2 = "SELECT recipient_code FROM recipient WHERE longitude = %s AND latitude = %s"
        cursor.execute(query2, (row['longitude'], row['latitude']))
        recipient_id = cursor.fetchone()[0]
        if row['tracking_number'] not in tracking:
            cursor.execute(
                '''INSERT INTO package (tracking_number, client_code, recipient_code,
                package_type, package_value, collect_amount, weight)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
                ON CONFLICT (tracking_number) DO NOTHING;
                ''', (row['tracking_number'], client_id, recipient_id, row['package_type'],
                      row['package_value'], row['collect_amount'], row['weight'])
            )
            tracking.add(row['tracking_number'])

    conn.commit()
    cursor.close()
```

```python
@task
def rds_del(df):
    hook = PostgresHook(postgres_conn_id='project_rds')
    conn = hook.get_conn()
    cursor = conn.cursor()

    del_id = 0
    df = df.drop_duplicates()
    for index, row in df.iterrows():
        del_id += 1
        query = "SELECT hub_id FROM hub WHERE delivery_hub = %s"
        cursor.execute(query, (row['delivery_hub'],))
        hub_id = cursor.fetchone()[0]
        query2 = "SELECT tracking_number FROM package WHERE tracking_number = %s"
        cursor.execute(query2, (row['tracking_number'],))
        track_id = cursor.fetchone()[0]
        cursor.execute(
                '''INSERT INTO delivery (delivery_id, tracking_number, hub_id, status, dispatch_date, delivery_date)
        VALUES (%s, %s, %s, %s, %s, %s)
        ON CONFLICT (delivery_id) DO NOTHING;
                ''', (del_id, track_id, hub_id, row['status'], row['dispatch_date'], row['delivery_date'])
            )

    conn.commit()
    cursor.close()

@task
def export_rds():
    postgres_hook = PostgresHook(postgres_conn_id='project_rds')
    s3_bucket = 'project-de-2023-jverdan'
    aws_credentials = BaseHook.get_connection('project_iam')

    s3_client = boto3.client(
    's3',
    aws_access_key_id=aws_credentials.login,
    aws_secret_access_key=aws_credentials.password
    )

    for i in ['hub', 'client', 'recipient', 'package', 'delivery']:
        sql_query = f"SELECT * FROM {i};"
        df = postgres_hook.get_pandas_df(sql_query)
        s3_key = f'gold/oltp_{i}.csv'
        df.to_csv(f'/tmp/oltp_{i}.csv', index=False, encoding='utf-8', sep=',')
        s3_client.upload_file(f'/tmp/oltp_{i}.csv', s3_bucket, s3_key)
```

```python
@task
def rs_dim(df):
    hook = RedshiftSQLHook(redshift_conn_id='project-rs')
    conn = hook.get_conn()
    cursor = conn.cursor()

    query = '''SELECT hub_id
            FROM dim_hub_info
            ORDER BY hub_id DESC
            LIMIT 1;'''
    try:
        hub_id = int(cursor.execute(query).fetchone()[0])
    except:
        hub_id = 0

    cursor.execute('SELECT DISTINCT delivery_hub FROM dim_hub_info')
    hubs = set(row[0] for row in cursor.fetchall())

    cursor.execute('SELECT DISTINCT date_id FROM dim_date')
    dates = set(row[0] for row in cursor.fetchall())

    for index, row in df.iterrows():
        if row['delivery_hub'] not in hubs:
            hub_id += 1
            cursor.execute(
                '''INSERT INTO dim_hub_info (hub_id, delivery_hub)
                VALUES (%s, %s)
                ''', (hub_id, row['delivery_hub'])
            )
            hubs.add(row['delivery_hub'])

        date_id = int(row['dispatch_date'].replace('/', ''))
        if date_id not in dates:
            date_format = '%m/%d/%Y'
            parsed_date = dt.strptime(row['dispatch_date'], date_format)

            # Extract the components
            month = parsed_date.month
            day = parsed_date.day
            year = parsed_date.year

         holiday_list = ['9/1/2023']
         sale_list = ['9/11/2023']

         if row['dispatch_date'] in holiday_list:
             is_holiday = True
         else:
             is_holiday = False

         if row['dispatch_date'] in sale_list:
             is_sale = True
         else:
             is_sale = False

         cursor.execute(
             '''INSERT INTO dim_date (date_id, year, month, day, is_holiday, is_ecommerce_sale)
             VALUES (%s, %s, %s, %s, %s, %s)
             ''', (date_id, year, month, day, is_holiday, is_sale)
         )
         dates.add(date_id)


   conn.commit()
   cursor.close()
```

```
@task
def rs_fact(df):
    hook = RedshiftSQLHook(redshift_conn_id='project-rs')
    conn = hook.get_conn()
    cursor = conn.cursor()

    df1 = df.groupby(['delivery_hub',
                      'dispatch_date']).agg(tracking_number_count=('tracking_number', 'count'),
                                            weight_sum=('weight', 'sum')).reset_index()
    cursor.execute('DELETE FROM fact_deliveries;')

    for index, row in df1.iterrows():
        query1 = 'SELECT hub_id FROM dim_hub_info WHERE delivery_hub = %s;'
        hub_id = int(cursor.execute(query1, (row['delivery_hub'],)).fetchone()[0])
        query2 = "SELECT date_id FROM dim_date WHERE date_id = %s;"
        date = int(row['dispatch_date'].replace('/', ''))
        date_id = cursor.execute(query2, (date,)).fetchone()[0]

        cursor.execute(
                '''INSERT INTO fact_deliveries (delivery_id, hub_id, date_id, package_count, package_weight)
                VALUES (%s, %s, %s, %s, %s)
                ''', (index, hub_id, date_id, row['tracking_number_count'], row['weight_sum'])
            )

    conn.commit()
    cursor.close()

@task
def export_rs():
    redshift_hook = RedshiftSQLHook(redshift_conn_id='project-rs')
    s3_bucket = 'project-de-2023-jverdan'
    aws_credentials = BaseHook.get_connection('project_iam')

    s3_client = boto3.client(
    's3',
    aws_access_key_id=aws_credentials.login,
    aws_secret_access_key=aws_credentials.password
    )

    for i in ['dim_hub_info', 'dim_date', 'fact_deliveries']:
        sql_query = f"SELECT * FROM {i};"
        df = redshift_hook.get_pandas_df(sql_query)
        s3_key = f'gold/olap_{i}.csv'
        df.to_csv(f'/tmp/olap_{i}.csv', index=False, encoding='utf-8', sep=',')
        s3_client.upload_file(f'/tmp/olap_{i}.csv', s3_bucket, s3_key)

data = read_csv_from_s3()
rds_hub(data) >> rds_package(data) >> rds_del(data) >> export_rds()
rs_dim(data) >> rs_fact(data) >> export_rs()


package()
```

Appendix E-3. DynamoDB ETL Code

```
import datetime
from datetime import datetime as dt
import pandas as pd
import boto3
from io import BytesIO
from airflow.decorators import dag, task
from airflow import DAG, Dataset
from airflow.operators.bash import BashOperator
from airflow.hooks.base import BaseHook
import pyarrow
import json


@dag(
    start_date=datetime.datetime(2023, 9, 15),
    schedule="@daily"
)

def rider():
    @task
    def read_csv_from_s3():
        s3_bucket = 'project-de-2023-jverdan'
        s3_key = 'landing/rider_raw_data.csv'

        aws_credentials = BaseHook.get_connection('project_iam')

        # Create an S3 client
        s3_client = boto3.client(
        's3',
        aws_access_key_id=aws_credentials.login,
        aws_secret_access_key=aws_credentials.password
    )

        # Use the S3 client to download the file
        response = s3_client.get_object(Bucket=s3_bucket, Key=s3_key)
        csv_content = response['Body'].read()

        # Load CSV data into a Pandas DataFrame
        df_raw = pd.read_csv(BytesIO(csv_content))
        return df_raw
```

```python
@task
def rider_data(df):
    aws_credentials = BaseHook.get_connection('project_iam')
    dynamodb = boto3.resource('dynamodb',
                              aws_access_key_id=aws_credentials.login,
                              aws_secret_access_key=aws_credentials.password)
    rider = dynamodb.Table('rider_data')
    rider2 = dynamodb.Table('rider_data2')

    client = boto3.client('dynamodb',
                              aws_access_key_id=aws_credentials.login,
                              aws_secret_access_key=aws_credentials.password)

    rider_id = client.execute_statement(
        Statement = """
        SELECT rider_id FROM rider_data
        """
    )['Items']

    if len(rider_id) != 0:
        values = [item['rider_id']['N'] for item in rider_id]
        riders = [int(value) for value in values]
    else:
        riders = []

    for index, row in df.iterrows():
        if row['rider_id'] not in riders:
            rider.put_item(Item={
                'rider_id': row['rider_id'],
                'first_name': row['first_name'],
                'last_name': row['last_name'],
                'driver_license_num': row['driver_license_num'],
                'tpsp_name': row['tpsp_name'],
                'date_started': row['date_started'],
                'hub_assignment': row['hub_assignment'],
            })

            rider2.put_item(Item={
                'rider_id': row['rider_id'],
                'tpsp_name': row['tpsp_name'],
                'date_started': row['date_started'],
                'hub_assignment': row['hub_assignment'],
            })
```

```
    @task
    def export_to_s3():
        aws_credentials = BaseHook.get_connection('project_iam')
        dynamodb = boto3.client('dynamodb',
                                aws_access_key_id=aws_credentials.login,
                                aws_secret_access_key=aws_credentials.password)

        s3_client = boto3.client(
        's3',
        aws_access_key_id=aws_credentials.login,
        aws_secret_access_key=aws_credentials.password
        )

        s3_bucket_name = 'project-de-2023-jverdan'
        s3_key = f'sensitive/rider_data.json'

        data = dynamodb.execute_statement(
                    Statement = """
                    SELECT * FROM rider_data
                    """
                )['Items']
        json_data = json.dumps(data, indent=2)

        # Upload JSON data to S3
        s3_client.put_object(Bucket=s3_bucket_name, Key=s3_key, Body=json_data)

        s3_key2 = f'gold/rider_data2.json'

        data = dynamodb.execute_statement(
                    Statement = """
                    SELECT * FROM rider_data2
                    """
                )['Items']
        json_data2 = json.dumps(data, indent=2)

        # Upload JSON data to S3
        s3_client.put_object(Bucket=s3_bucket_name, Key=s3_key2, Body=json_data2)

    data = read_csv_from_s3()
    rider_data(data) >> export_to_s3()

rider()
```